FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Processador Aritmético com Arquitetura a Fluxo de Dados

RAFAEL GASPERETTI

MARÍLIA, 2014

FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Processador Aritmético com Arquitetura a Fluxo de Dados

Monografia do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador

Profº Ms. Ildeberto de Genova Bugatti

MARÍLIA, 2014

"Na natureza nada se cria, nada se perde, tudo se transforma".

Antoine Laurent de Lavoisier

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Rafael Gasperetti

Processador Aritmético com Arquitetura a Fluxo de Dados

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10 (dez)

Orientador: Ildeberto de Gênova Bugatti

1°. Examinador: Paulo Augusto Nardi

2°. Examinador: Fábio Dacêncio Pereira

Marília, 02 de dezembro de 2014.

GASPERETTI, R.. **Processador Aritmético com Arquitetura a Fluxo de Dados**. 2014. XX f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

RESUMO

Máquinas com arquitetura à Fluxo de Dados são organizadas de forma a possibilitar alto grau de paralelismo. O mecanismo de execução de instruções numa máquina a fluxo de dados é realizado de forma diferenciada do mecanismo de execução de instruções em máquinas que possuem arquitetura de Von Neumann.

Existe uma diversidade de arquiteturas paralelas contendo arranjos de processadores, no entanto, geralmente a arquitetura dos processadores contidos nessas máquinas possui arquitetura de Von Neumann. Nessas máquinas, a sequência de execução de instruções é definida pelo programador e, na arquitetura a fluxo de dados a sequência de execução de instruções é definida pela disponibilidade de dados, assim, essas arquiteturas também são denominadas arquiteturas dirigidas por dados (Data Driven).

Na arquitetura a fluxo de dados a execução de instruções de forma simultânea é definida em tempo de execução, sem interferência do programador. Nas arquiteturas paralelas com processadores Von Neumann a obtenção de paralelismo precisa ser identificada ou definida pelo programador, gerando alto grau de ociosidade dos processadores contidos nessas máquinas.

Esse projeto propõe a construção de um processador aritmético com arquitetura a fluxo de dados, visando explorar as características de execução de instruções dessa arquitetura para obter alto grau de paralelismo na execução de operações contidas em expressões aritméticas.

Palavras-chave: Fluxo de Dados, Dirigido por Dados, Paralelismo, Multiprocessamento, Pipeline, Processador Aritmético.

GASPERETTI, R.. **Arquiteturas Computacionais Paralelas Específicas**. 2014. XX f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

ABSTRACT

Machines with data flow architecture are organized to allow high degree of parallelism. The execution mechanism of the instructions on a data stream machine are performed differently from that with Von Neumann's architecture.

There are a lot of architectures containing processors parallel arrays, however, generally the processors architecture contained in these machines has Von Neumann architecture. In these machines, the programmer defines the execution sequence of the instructions but in data flow, architectures are defined by the data availability, that is why they are also called data driven architectures.

In data flow architecture, the simultaneously execution of the instructions are defined at runtime, without the programmer interference. In parallel architectures with Von Neumann processors, obtain parallelism requires to be identified or defined by the programmer, generating high degree of idleness on the processors contained on these machines.

This project proposes the construction of an arithmetic processor with a data flow architecture, aiming to explore the characteristics of instruction execution of this architecture to obtain high degree of parallelism in the execution of operations contained in arithmetic expressions.

Keywords: data flow, Data Driven, Parallelism, Multiprocessing, Pipeline, Arithmetic Processor.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AP: Atomic Processor;

ASU: Atomic Storage Unit;

E/S: Entrada e Saída;

FIFO: First In First Out;

FPGA: Field Programmable Gate Array;

Hz: Hertz;

K: Kilo

LAU: Linguagem de Atribuição Única;

LIFO: Last In First Out;

M: Mega;

MIT: Massachusetts Institute of Technology;

RAM: Random Access Memory;

RGB: Red, Green, Blue;

ULA: Unidade Lógica e Aritmética;

VGA: Video Graphics Array;

VHDL: VHSIC Hardware Description Language;

VHSIC: Very High Speed Integrated Circuits;

VLSI: Very Large Scale Integration.

ÍNDICE

1. Introdução e objetivos

Este capítulo tem a funcionalidade de introduzir e contextualizar o tema do projeto para facilitar a compreensão acerca do seu desenvolvimento.

1.1. Introdução

Com a evolução da computação como um todo, tanto em nível de hardware como em nível de software, viabilizou-se a utilização de arquiteturas executando cálculos de forma paralela e simultânea, pois deste modo, o conjunto de atividades ou sequência de instruções podem ser executadas de forma mais eficiente, ou seja, com menor tempo de resposta (Bugatti, Ildeberto de Genova, 1986).

A real execução paralela de instruções nessas arquiteturas é possibilitada devido a existência de mais de um núcleo de processamento, gerando processamento paralelo, que executa mais de uma instrução de forma simultânea, com o objetivo de diminuir o tempo de execução de um programa. Cada núcleo pode executar instruções diferentes de um mesmo programa ou de programas diferentes, utilizando critérios e técnicas de programação e processamento a Fluxo de Dados para distribuir a carga de processamento entre os núcleos de forma homogênea e balanceada (Bugatti, Ildeberto de Genova, 1986).

Máquinas com arquitetura à Fluxo de Dados apresentam um mecanismo de controle de execução de instruções que difere do mecanismo de execução de instruções de arquiteturas convencionais, como a arquitetura de Von Neumann ou Harvard. Nas máquinas com arquitetura a fluxo de dados, a sequência de execução de instruções é definida pela disponibilidade de operandos, diferenciando das máquinas com arquiteturas tradicionais, onde a sequência de execução de instruções é definida pelo programador. Assim, em uma arquitetura a fluxo de dados a sequência ou ordem de execução de instruções de um programa é dirigida por dados (Data Driven) (Bugatti, Ildeberto de Genova, 1986).

O processador aritmético com arquitetura a fluxo de dados proposto foi implementado utilizando tecnologia de componentes eletrônicos programáveis do tipo FPGA e linguagem de descrição de hardware (VHDL).

1.2. Objetivos

O desenvolvimento do trabalho propões dois objetivos principais. O primeiro objetivo é sintetizar, implementar e validar técnicas de detecção de paralelismo em programas utilizando uma máquina com arquitetura a Fluxo de Dados utilizando múltiplos núcleos de processamento com pipeline entre os módulos. O segundo objetivo é utilizar técnicas de programação à Fluxo de Dados (*Data Flow*) ou Dirigida por Dados (*Data Driven*) para detectar paralelismo em expressões aritméticas em tempo de execução.

Vale ressaltar que a detecção do paralelismo neste projeto restringe-se à detecção de paralelismo em expressões aritméticas para adequar o tamanho do projeto com o tempo de desenvolvimento.

O processador aritmético com arquitetura a Fluxo de Dados é obtido assim que concluído o desenvolvimento de tais objetivos.

O texto da monografia descreve inicialmente os objetivos do trabalho, seguido da descrição de todas atividades realizadas na execução do projeto para contemplar os objetivos iniciais. O capítulo 2 contém os conceitos sobre: programação; mecanismo de execução de instruções a fluxo de dados, incluindo a relação e descrição das principais características dos computadores com arquitetura a fluxo de dados estudados. O capítulo 3 descreve a síntese, projeto e implementação do processador aritmético com arquitetura a fluxo de dados utilizando tecnologia de componentes eletrônicos programáveis (FPGA) e descreve os testes e validação do processador construído utilizando simulações de execução das expressões aritméticas. O quarto capítulo contém conclusões sobre os resultados obtidos e sugestões de continuidade. O capítulo 5 faz referência sobre a bibliografia utilizada no projeto.

2. Contextualização sobre fluxo de dados e tecnologias pesquisadas

Neste capítulo são descritos conceitos sobre: programação a fluxo de dados e mecanismo de execução de instruções (item 2.1), grafos a fluxo de dados (item 2.2) e arquiteturas a fluxo de dados pesquisadas (item 2.4) (Bugatti, Ildeberto de Genova, 1986);

O item 2.1, relaciona e descreve conceitos sobre programação a fluxo de dados e o mecanismo de execução de instruções. O item 2.3, relaciona e descreve as principais características de grafos a fluxo de dados e exemplos de programas em comparação com os grafos.

2.1. Programação a fluxo de dados e mecanismo de execução de instruções

Um programa a fluxo de dados pode ser representado por um grafo, onde os nós representam operadores que são interconectadas por enlaces, através dos quais fluem valores, denominado grafo a fluxo de dados. A figura 1 descreve um grafo a fluxo de dados representando a expressão aritmética "$X = (A + B) * (A + C) * (B + C)$", onde o operador identificado por "C" representa o operador cópia, explicado no item 2.2.4.

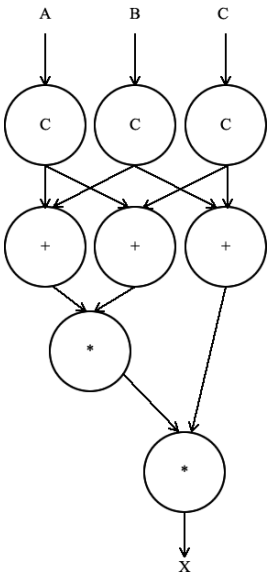

Figura 1: Execução de uma expressão aritmética em fluxo de dados.

Um operador a fluxo de dados é habilitado quando valores estiverem presentes em todos os seus enlaces de entrada. O operador é habilitado para disparar (iniciar a execução) a qualquer instante, removendo os valores de seus enlaces de entrada, computando esses valores e enviando um resultado para o seu enlace de saída. Um operador não pode ser executado (disparado), caso exista um dado em algum arco de saída do operador. Um resultado pode ser enviado a mais de um destino através da utilização de uma operação cópia (descrito no item 2.2.4), que remove um valor de seu enlace de entrada e coloca o valor em seus enlaces de saída, produzindo assim, cópias dos valores de entrada.

No grafo a fluxo de dados as operações que podem ser executadas de forma simultânea (paralela) são identificados de forma eficiente e intuitiva.

Vale ressaltar que arquiteturas à Fluxo de Dados apresentam um mecanismo de controle de execução de instruções que difere do mecanismo de execução de instruções de arquiteturas convencionais, com arquiteturas de Von Neumann ou Harvard. Nas máquinas a fluxo de dados, a sequência de execução de instruções é gerada pela disponibilidade de operandos, diferenciando das máquinas com arquiteturas tradicionais, onde a sequência de execução de instruções é definida pelo programador. Assim, a sequência ou ordem de execução de instruções de um programa é dirigida por dados (Data Driven). Exemplos de grafos a fluxo de dados são descritos no item 2.3.

2.2. Principais Operadores Contidos em uma Linguagem à Fluxo de Dados

Todo operador a fluxo de dados possui enlaces de entrada e saída e especifica uma operação entre os valores dos dados contidos ou disponibilizados em seus enlaces de entrada. O resultado gerado é inserido em seu enlace de saída. Todas as operações são locais para os enlaces de um único operador; operações não possuem efeitos secundários. Cada enlace de dados conecta a saída de um operador a entrada de outro operador; enlaces especificam dependências de dados entre as instruções de um programa.

Um programa escrito em uma linguagem a fluxo de dados pode ser representado por um grafo dirigido. Em uma linguagem de programação tradicional, um programa genérico é constituído por quatro famílias de instruções: instruções lógicas aritméticas; instruções de

movimentação de dados, instruções de controle e instruções de entrada e saída. Em uma linguagem fluxo de dados, além dos operadores lógicos e aritméticos também são necessários operadores de controle. Nesse trabalho, iremos estudar e definir essas duas primeiras famílias, pois o objetivo do mesmo é construir um processador aritmético com arquitetura a fluxo de dados.

Os itens que seguem descrevem as famílias de operadores a fluxo de dados existentes em grafos de fluxo de dados. São eles operadores lógicos, aritméticos, relacionais e de controle.

2.2.1. Operadores lógicos

São operadores que utilizam apenas um bit para retornar um resultado, tais como AND, OR, NOR, XOR e NOT. O processador aritmético construído contém todos operadores lógicos contido na figura 2.

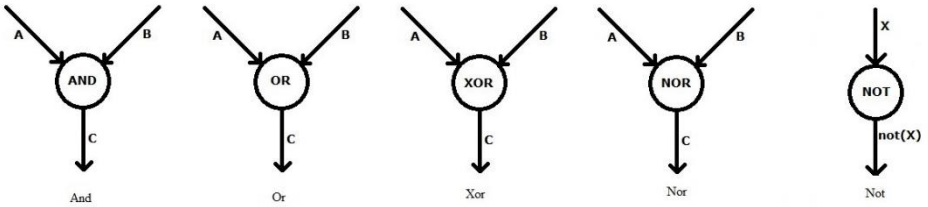

Figura 2: Operadores Lógicos.

2.2.2. Operadores aritméticos

São operadores que realizam a operações aritméticas básicas: soma, subtração, multiplicação e divisão. O processador aritmético implementado contém todas as operações relacionadas e utilizam todos os bits da arquitetura para retornar um resultado. A figura 3 mostra a representação dos operadores a fluxo de dados aritméticos.

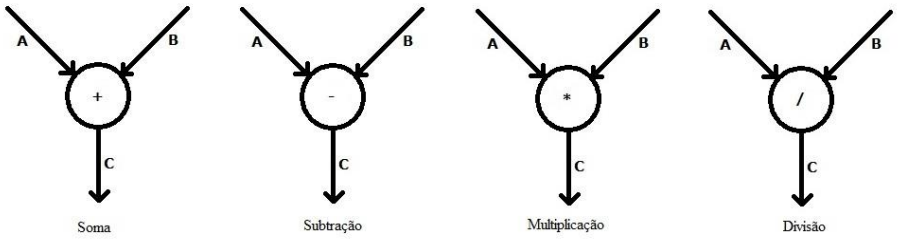

Figura 3: Operadores Aritméticos.

2.2.3. Operadores relacionais

No projeto do processador aritmético foram implementadas todas as operações relacionais: igual, diferente, maior, menor, maior ou igual e menor ou igual. A figura 4 mostra a representação gráfica dos operadores relacionais.

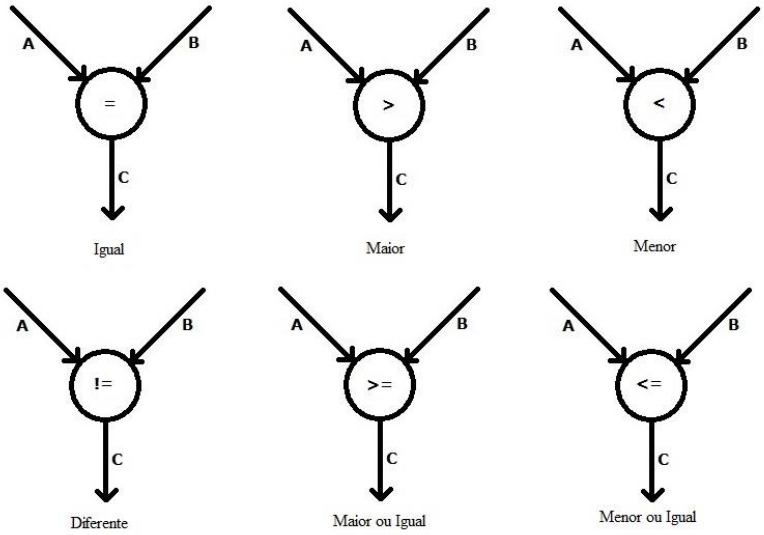

Figura 4: Operadores Relacionais.

2.2.4. Operadores de controle

O conjunto de operadores de controle proposto, possibilita a construção de estruturas de decisão e comandos iterativos. Esses operadores permitem a geração de cláusulas condicionais e estruturas de repetição na linguagem de programação a fluxo de dados utilizada.

Os operadores de controle propostos são denominados: comutação, união e cópia. A figura 5 mostra a forma gráfica dos operadores de controle propostos. O texto que segue descreve a forma de funcionamento dos mesmos.

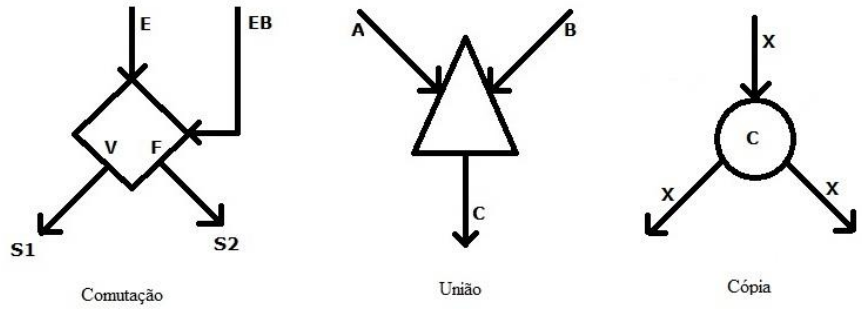

Figura 5: Operadores de Controle.

O operador Comutação possui dois enlaces de entrada denominados "*E*" e "*EB*", e dois enlaces de saída, denominados "*S1*" e "*S2*". Ele transfere o dado disponibilizado em seu enlace de entrada "*E*" para apenas um dos enlaces de saída, dependendo do valor lógico do dado disponibilizado no enlace de entrada EB. Gerando a seguinte estrutura:

Se EB = TRUE
Então S1=E
Senão S2=E

O operador de união contém dois enlaces de entrada e um enlace de saída. Ele transfere para o seu enlace de saída o dado contido em apenas um dos valores contidos em um dos seus enlaces de entrada. O dado transferido é aquele que acontece primeiro. A figura 5 mostra a forma gráfica dos operadores de controle propostos.

O operador cópia recebe um valor em seu enlace de entrada e copia este valor em seus dois enlaces de saída, permitindo assim, a utilização em mais de uma operação ao mesmo tempo.

A existência do operador de comutação possibilita a construção de repetições e tomadas de decisão.

2.3. Grafos e exemplos de programas a fluxo de dados

Na arquitetura que será desenvolvida, serão utilizadas as operações aritméticas básicas (soma, subtração, multiplicação e divisão). O exemplo da figura 1 exibida no item 2.1 mostra a execução de um programa a Fluxo de Dados que ilustra o mecanismo de execução da seguinte expressão aritmética:

$$\text{“}X = (A + B) * (A + C) * (B + C)\text{”}$$

A mesma resolução em alto nível, utilizando pseudo código, é solucionada da seguinte maneira:

$begin$

$int\ a = 2, b = 3, c = 4, x;$

$x = (a + b) * (a + c) * (b + c);$

$return\ x;$

end

A figura 6 mostra execução de um programa que traz como o resultado o módulo de determinado valor, a operação ABS(A).

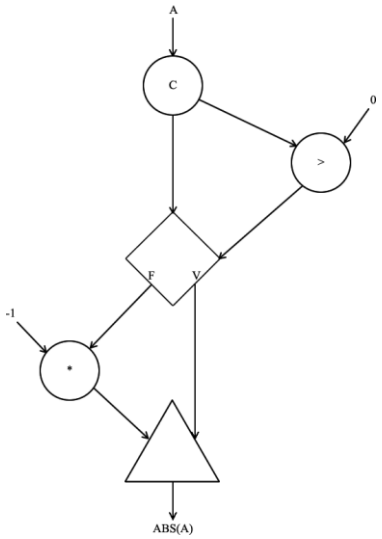

Figura 6: Execução da expressão ABS(A) em Fluxo de Dados.

A mesma resolução em alto nível, utilizando pseudo código, é solucionada da seguinte maneira supondo-se que o valor de "a" seja um valor do tipo inteiro:

$begin$

$int\ a = numeroInteiro();$

$if(a > 0){$

$return\ a;$

$}else{$

$return\ a * (-1);$

$}$

end

A figura 7 mostra a execução de um programa que executa repetidamente uma parte dele até que uma condição booleana seja satisfeita.

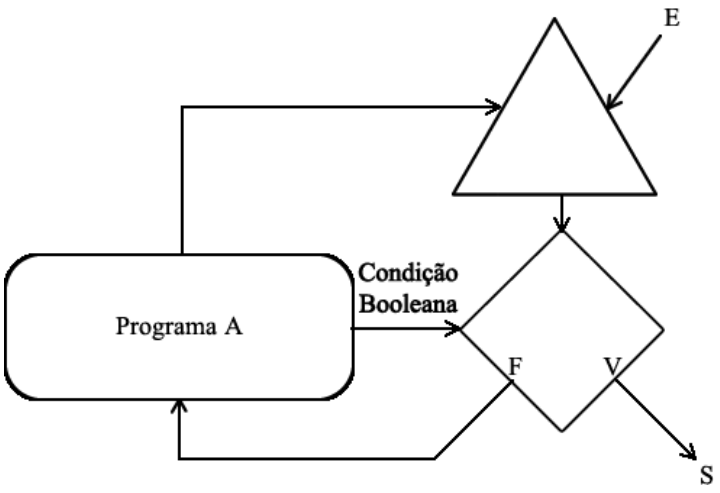

Figura 7: Grafo Dirigido em uma Estrutura de Repetição.

A mesma resolução em alto nível, utilizando pseudo código, é solucionada da seguinte maneira:

$begin$

$S = E;$

$boolean\ condicao = false;$

$While\ (condicao == false)\{$

$condicao = ProgramaA(S);$

$\}$

$return\ S;$

end

Vale ressaltar que a arquitetura é capaz de, em tempo de execução, detectar a dependência de dados e executar todas as operações possíveis ao mesmo tempo, reduzindo assim, o tempo de execução do conjunto de instruções, computador a fluxo de dados do instituto de tecnologia de Massachusets (MIT), máquina dirigida por dados de Utah, máquina a fluxo de dados de Irvine, computador a fluxo de dados de Manchester, sistema LAU de Toulouse e computador a fluxo de dados de Newcastle.

2.4. Relação e descrição das arquiteturas a fluxo de dados estudadas

Para um melhor desenvolvimento deste projeto, foram estudadas várias arquiteturas a Fluxo de Dados, para entre elas, escolher uma para implementação. São elas: Mecanismo básico de execução de instrução em máquinas a fluxo de dados,

2.4.1. Mecanismo básico de execução de instrução em máquinas a fluxo de dados

A arquitetura gerada nesse item é utilizada para descrever o funcionamento básico de uma arquitetura a Fluxo de Dados, assim como definir as funções dos principais subsistemas utilizados e necessários em máquinas com arquitetura a fluxo de dados.

A figura 8, mostra em diagrama de blocos, os principais módulos contidos em uma arquitetura a fluxo de dados básica denominados: Unidade Operação, Unidade de Atualização,

Pilha de Instruções, Unidade e Armazenamento de Atividades e Unidade de Busca. A organização dos módulos possibilita troca de informações entre módulos contíguos. Essa forma caracteriza uma arquitetura a fluxo de dados pipeline. Onde cada um dos módulos pode executar suas atividades de forma também simultânea, caracterizando um paralelismo pipeline. A figura 8 é utilizada para descrever o mecanismo de execução de instruções na arquitetura a fluxo de dados.

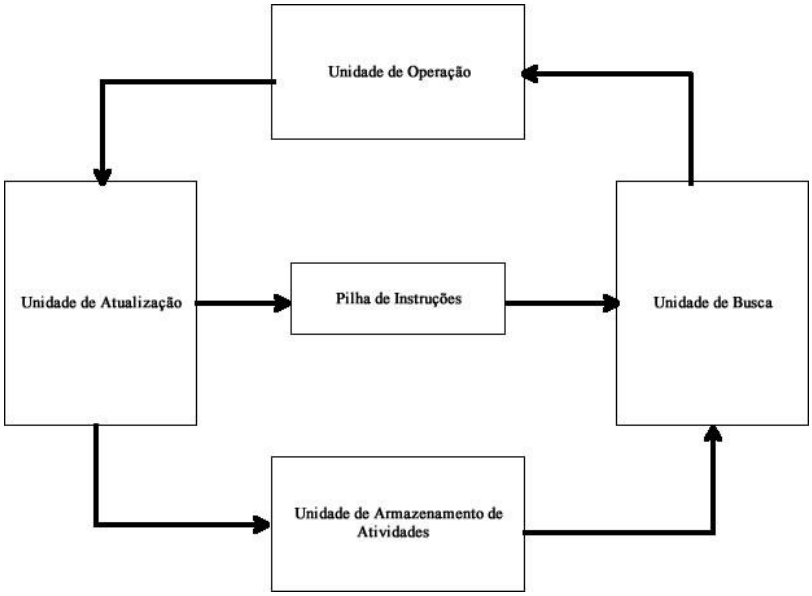

Figura 8: Mecanismo básico de execução de instrução.

Um programa Fluxo de Dados é formado por um conjunto de instruções que estão armazenadas na unidade de armazenamento de atividades. Cada instrução possui um endereço único na pilha de instruções (LIFO). Assim que estiver apta a execução (recebido todos os operandos) a Unidade de Organização localiza o endereço da instrução na pilha de instruções e lê o comando a ser executado na unidade de armazenamento de atividades, gerando um pacote de operação.

Um pacote de operação é um conjunto de dados a ser executado. O pacote de operação deve conter ao menos os operandos necessários para uma operação, o código da operação que será realizada na Unidade de Operação, para que o resultado seja o desejado, além do endereço de destino do resultado, para que o resultado obtido seja utilizado na próxima operação. A tabela

1 que segue mostra um pacote básico de operação utilizando valores de 8 bits para representar uma operação a ser realizada.

Tabela 1: Pacote básico de operação.

Operando A	Operação	Operando B	Endereço Destino
10101010	00000000	10110100	10010010

O pacote de operação é enviado para a unidade de operação. A unidade de operação realiza a atividade correspondente ao código de operação sobre os valores (operandos), gerando um pacote de dado para cada campo de destino do pacote de operação. A Unidade de Distribuição de Dados recebe os pacotes de resultados e introduz os valores contidos neles nos campos de operandos (receptores) da instrução especificada pelo seu campo de destino. É papel da Unidade de Distribuição de Dados, também, reconhecer se todos os operandos necessários para disparar a instrução foram recebidos, em caso positivo, introduz o endereço da instrução na pilha de instruções.

Para medir o grau de concorrência em um programa durante sua execução, basta verificar o número de entradas na pilha de instrução. A arquitetura contida na figura 8 utiliza este potencial significantemente. Assim que a unidade de busca envia um pacote para a unidade de operação, já recebe outro endereço da pilha sem esperar o término da instrução anterior. Assim, uma cadeia contínua de pacotes de operação passa da unidade de busca para a unidade de operação enquanto a pilha não estiver vazia.

Esta passagem de pacotes pode ser feita simultaneamente em diferentes partes do anel (pipeline), sendo assim, as diferentes instruções são executadas concorrentemente, ou seja, o anel opera em "pipeline" com todas as unidades, processando ativamente uma instrução de cada vez em cada unidade lógica e aritmética.

O limite do grau de concorrência é dado pelo número de unidades do anel e pelo grau do paralelismo em cada unidade. Para aumentar a concorrência pode-se desdobrar uma unidade do anel em várias, utilizando-as para alocar atividades concorrentes, ou então conectar vários elementos de processamento (PE) em paralelo, utilizando um sistema de multiprocessamento de Fluxo de Dados, e assim, elevando significativamente o grau de concorrência.

2.4.2. Computador de fluxo de dados do Instituto de Tecnologia de Massachusets (MIT)

A figura 9 mostra, em diagrama de blocos, a arquitetura do computador de Fluxo de Dados do MIT, que gerou uma contribuição significativa para as pesquisas de computadores em Fluxo de Dados.

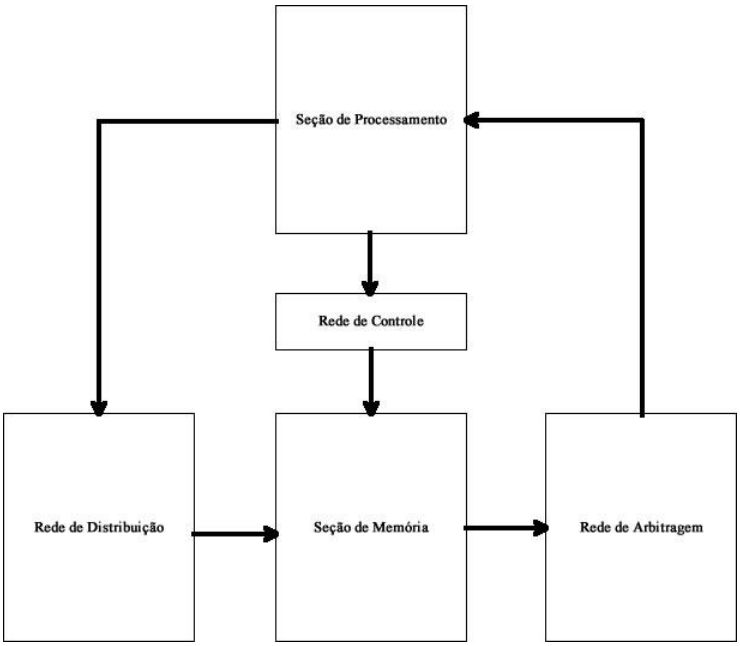

Figura 9: Computador de Fluxo de Dados do Instituto de Tecnologia de Massachusets (MIT).

Na máquina do MIT os dados caminham assincronamente pelos canais de comunicação que interligam os módulos.

- A **seção de memória** contém o conjunto de instruções e os operandos;

- A **seção de processamento** contém os processadores, que executam das instruções requisitadas de acordo com o código de cada uma;

- A **rede de arbitragem** encaminha o conjunto de instruções da seção de memória para a seção de processamento;

- **A rede de controle** encaminha os pacotes de controle da seção de processamento para a seção de memória;

- A **rede de distribuição** encaminha os pacotes de dados da seção de processamento para a seção de memória.

2.4.3. Máquina dirigida por dados de Utah

Esta arquitetura possui uma estrutura recursiva, conforme mostra a imagem contida na figura 10.

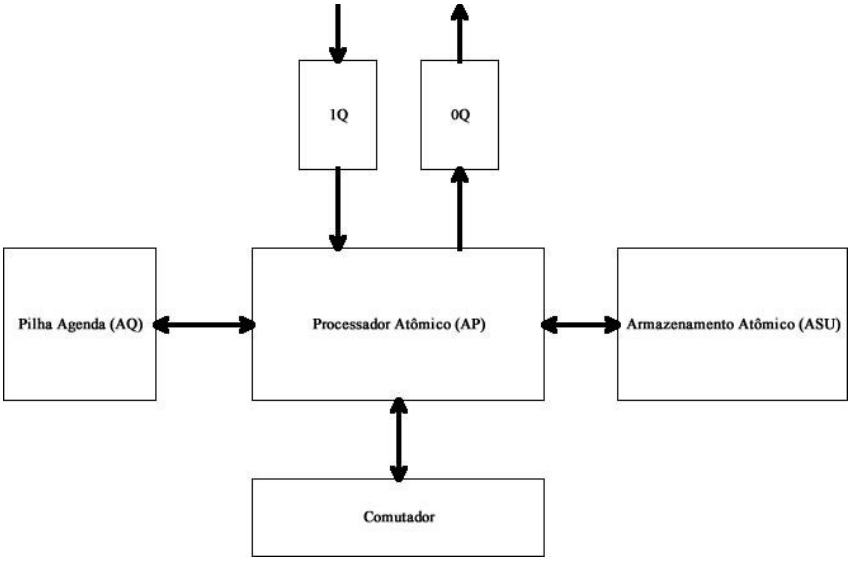

Figura 10: Máquina dirigida por dados de Utah.

A máquina mostrada na figura 10 contém seis módulos principais que interligados atuam de modo recursivo, são eles:

- **Unidade de Armazenamento Atômico (ASU):** módulo que contém a memória com as sequências de instruções do programa;

- **Processador Atômico (AP):** unidade responsável por executar as instruções do programa;

- **Pilha Agenda (AQ)**: as mensagens de armazenamento atômico são colocadas nesta pilha;

- **Pilha de Entrada (1Q)**: buffers de entrada de mensagem do elemento de computação superior;

- **Pilha de Saída (0Q)**: buffers de mensagem para o elemento de computação superior;

- **Comutador**: conecta os elementos de computação com os outros 8.

2.4.4. Máquina a fluxo de dados de Irvine

Esta máquina foi desenvolvida para explorar a tecnologia VLSI e o alto grau de concorrência entre os programas a serem executados.

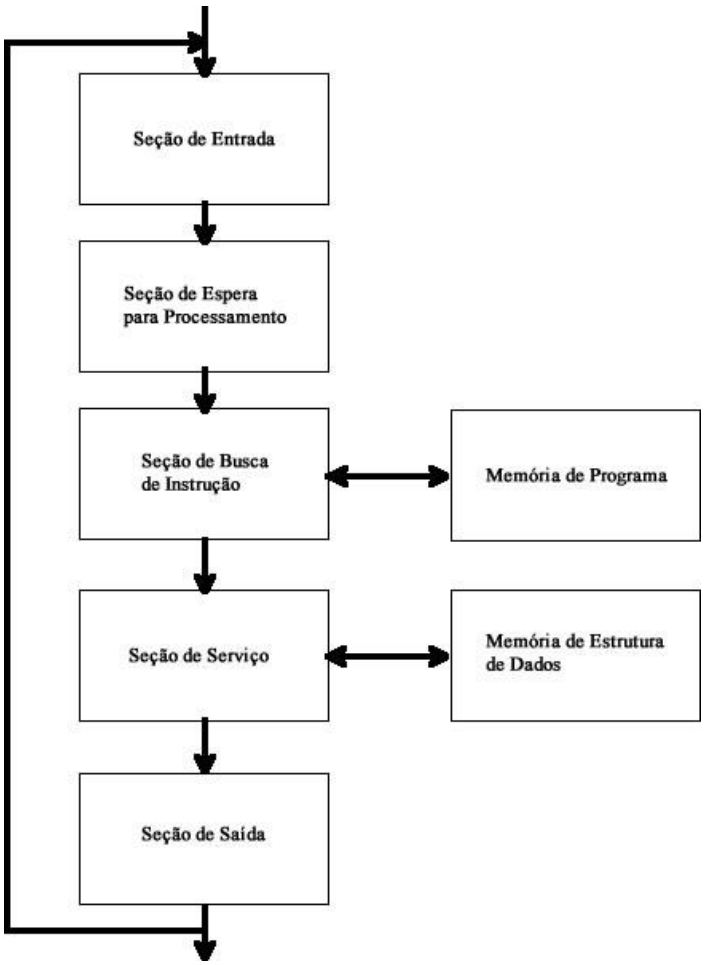

Figura 11: Máquina Fluxo de Dados Irvine.

Conforme a figura 11, cada módulo é essencialmente um computador completo, tendo o seu conjunto de instruções de 16K de palavras com o propósito de armazenar programas e estruturas de dados. Seus elementos são:

- **Seção de entrada**: tem o propósito de receber mensagens de outros elementos de processamento;

- **Seção de espera para processamento**: este módulo contém os pacotes de dados para as instruções consumidoras;

- **Seção de serviço**: unidade lógica aritmética (ULA) com ponto flutuante.

- **Seção de saída**: tem o propósito de enviar mensagens para outros elementos de processamento.

2.4.5. Computador a fluxo de dados de Manchester

Este projeto foi desenvolvido com o intuito da obtenção de um computador de alto desempenho. A figura 12 mostra a arquitetura a fluxo de dados da máquina em diagrama de blocos:

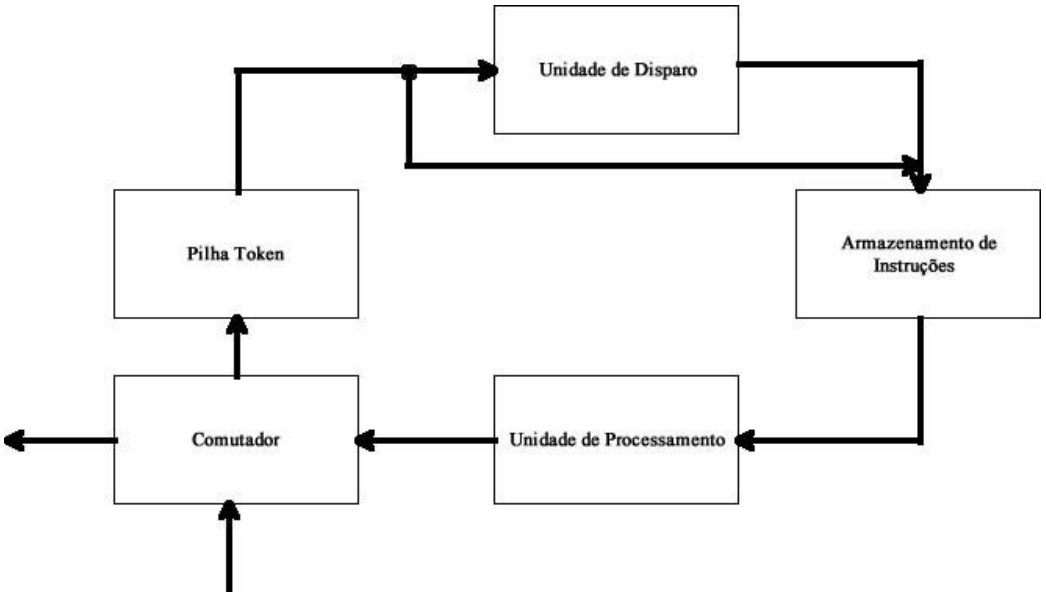

Figura 12: Computadores Fluxo de Dados Manchester.

Este computador é composto dos seguintes módulos:

- **Comutador**: atuador nas funções de entrada e saída (E/S);
- **Pilha Token**: encarregado de armazenar temporariamente um pacote de dado;
- **Unidade de disparo**: armazena as instruções que serão executadas;
- **Armazenamento de instruções**: onde ficam os programas Fluxo de Dados;
- **Unidade de processamento**: elemento que executa as instruções, podendo possuir vários elementos processadores idênticos atuando em paralelo. Para iniciar a execução de uma parte de um programa, pacotes de dados são inseridos na arquitetura através do módulo comutador. Ao final da execução são geradas instruções contendo pacotes de saída.

2.4.6. Sistema LAU de Toulouse

Este sistema é um computador dirigido por dados, criado para executar a "Linguagem de Atribuição Única" (LAU), idealizada e utilizada para resolver inúmeros problemas com Fluxo de Dados. Esquematiza-se esta arquitetura na figura 13:

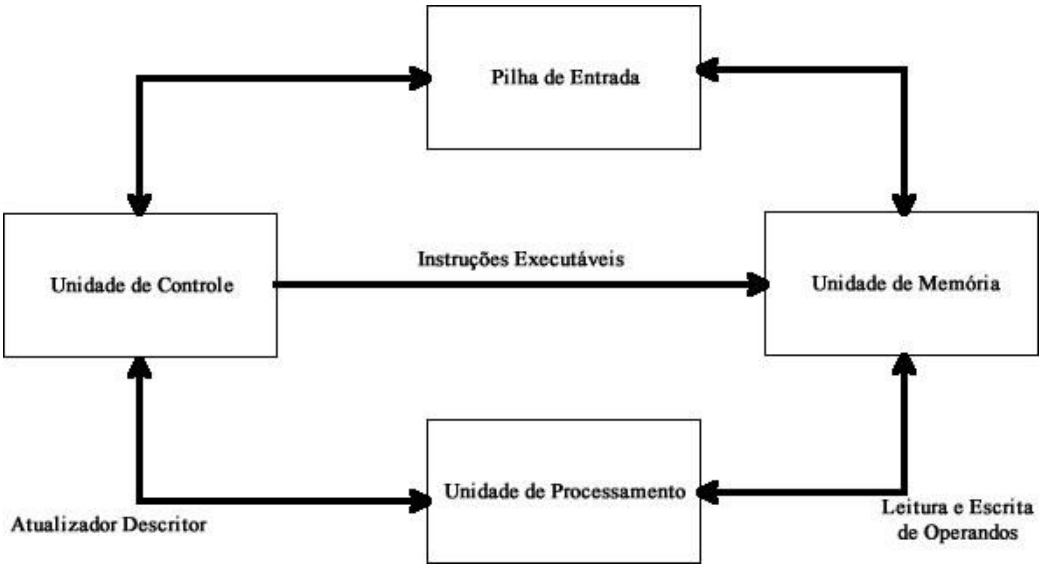

Figura 13: Sistema LAU de Toulouse.

Conforme o mostrado na figura 13, esta arquitetura é composta das seguintes unidades:

- **Pilha de Entrada:** Possui a responsabilidade de distribuir tanto instruções de controle quanto de dados;

- **Unidade de memória**: Responsável por armazenar os conjuntos de instruções;

- **Unidade de controle**: Módulo que realiza o controle da memória;

- **Unidade de processamento**: Possui 32 elementos idênticos de processamento. Este projeto tem uma especificidade interessante, onde o contador de programa, da arquitetura de Von Neumann, tem a sua função realizada por duas memórias nesta arquitetura.

2.4.7. Computador a fluxo de dados de Newcastle

Esta arquitetura é uma das primeiras máquinas a fluxo de dados proposta e encontrada no levantamento realizado sobre o tema. Ela é constituída pelas três estruturas básicas que caracterizam uma arquitetura a fluxo de dados e que possibilitam o mecanismo de execução a

fluxo de dados onde, a ordem de execução de instruções é determina pela disponibilidade de operandos. A figura 14 mostra, em diagrama de blocos, a arquitetura a fluxo de dados de Newcastle.

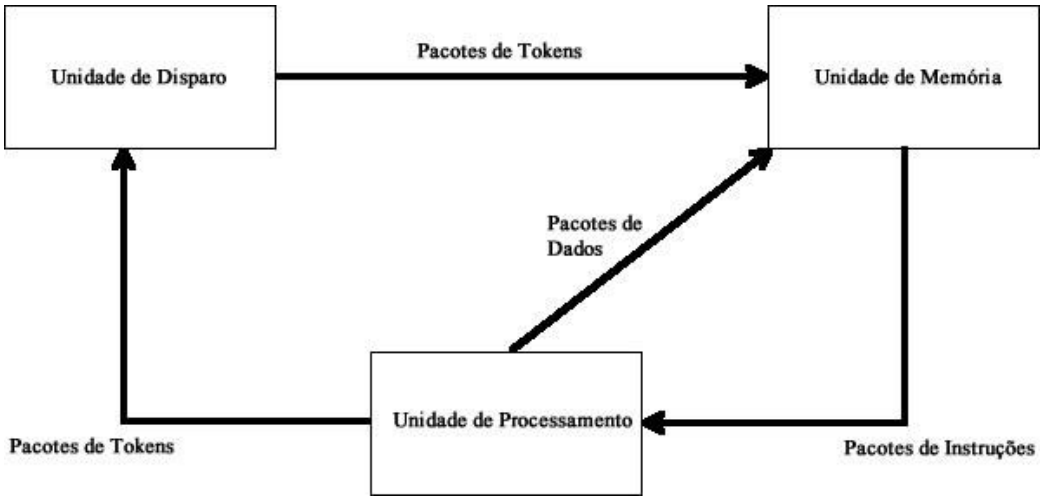

Figura 14: Computador Fluxo de Dados de Newcastle.

Conforme o mostrado na figura 14, esta arquitetura possui três módulos básicos:

- **Unidade de Disparo**: Responsável pela ativação das instruções;
- **Unidade de Memória**: Armazena-se na memória as instruções dos programas.
- **Unidade de Processamento**: Unidade responsável pela execução das instruções e distribuir os resultados.

Entre as arquiteturas estudadas, que melhor se encaixa para o desenrolar do projeto é o mecanismo básico de execução de instruções a fluxo de dados, pois contém apenas os módulos básicos necessários para um computador ser considerado a fluxo de dados, portanto permite maior versatilidade na adaptação de alguns módulos para as características necessárias no projeto.

3. Descrição, síntese, dimensionamento, construção e validação do processador

No capítulo anterior foram estudadas diversas arquiteturas a Fluxo de Dados, dentre as quais o mecanismo básico de execução de instruções a fluxo de dados foi selecionado para que o projeto seja baseado e desenvolvido. O capítulo atual descreve esta arquitetura com o intuito de explicar as técnicas utilizadas para desenvolvê-la e as validações necessárias para que as operações retornem o resultado esperado.

Para que o projeto fosse desenvolvido foi necessária a síntese, implementação, validação e utilização de quatro módulos principais utilizando a linguagem de descrição de hardware VHDL, que permite a criação e utilização de circuitos através de códigos de programação utilizando a sintaxe correta desta linguagem.

Os módulos da arquitetura implementada (Unidade de Memória e Ativação, Unidade de Instruções Ativadas, Unidade de Operação e Unidade de Atualização de Dados), visualizados em diagrama de blocos na figura 15, possuem características próprias, onde cada um deles executa uma parte do processo de execução de operações para a obtenção de um resultado final dentro do esperado.

A Unidade de Memória e Ativação (item 3.1.1) tem a função de armazenar todas as operações a serem executadas de diversos programas e dispará-las individualmente assim que cada uma delas estiver com todos os operandos disponíveis. Para a implementação foi necessária a utilização de marcadores que contém um total de oito bits para controlar a situação de cada endereço da memória, que juntos sinalizam o estado atual do endereço.

Os marcadores indicam características como a disponibilidade dos operandos, do operador e do endereço de destino, a quantidade de operandos que a operação possui, o operando de destino dentro do endereço de destino e o uso do endereço (livre ou ocupado), onde todas essas características necessárias são validadas sempre que um endereço é modificado para a execução da operação caso todos os operandos estejam disponíveis. Este módulo conta também com quatro núcleos de recepção e ativação para ser compatível com a Unidade de Operação (item 3.1.3) e não gerar gargalos durante a execução.

A Unidade de Memória e Ativação deve também verificar a lotação das filas de execução da Unidade de Operação e enviar cada operação para a fila com a menor quantidade de operações prontas para serem executadas, o que fez necessária a validação de disparos simultâneos para a mesma fila de execução evitando a perda de alguma operação em um único período de oscilação do relógio caso sejam ativadas ao mesmo tempo. Para tanto, foi necessária a implementação de prioridades para os núcleos de execução da Unidade de Memória e Ativação, onde o primeiro núcleo possui prioridade para acessar a menor fila em relação ao restante, o segundo núcleo possui prioridade em relação ao terceiro e ao quarto e o terceiro possui prioridade em relação ao quarto. Sempre que a prioridade é validada, aqueles que possuem menor prioridade utilizam filas com mais operações prontas para a execução.

A Unidade de Instruções Ativadas (item 3.1.2, página 36) também possui quatro núcleos de execução para compatibilizar com o restante da arquitetura. Este módulo é responsável por retirar a operação da fila de instruções prontas e enviar para a Unidade de Operação assim que esta finaliza a execução anterior. A verificação é permitida através de uma *flag* que a Unidade de Operação possui chamada "*exec*" com a finalidade de informar se está ou não em execução, garantindo que não haja ociosidade entre o fim de uma operação e o início de outra.

A Unidade de Operação (item 3.1.3) é responsável pela execução de todas as operações, para tanto foram necessárias as implementações de algoritmos que realizem operações como multiplicação, divisão, inverso de um valor, deslocamento para a direita e para esquerda, operações lógicas e operações relacionais, além das operações já pertencentes a sintaxe do VHDL (soma e subtração). Para utilizar a operação desejada foram criados identificadores únicos para cada operação de modo que ao inserir determinado valor, é executada determinada operação.

A Unidade de Atualização de Dados (item 3.1.4) tem o dever de receber o resultado da Unidade de Operação e atualizar a Unidade de Memória e Ativação com os novos valores, de modo que este resultado possa ser utilizado em uma operação futura se necessário, que será executada assim que todos os seus operandos estiverem disponíveis.

Vale ressaltar que o item 3.2.3 descreve o ambiente de testes utilizado juntamente com o hardware utilizado em tais testes (figura 18), destacando o componente FPGA de código Nexys-3^{TM} Spartan-6 desenvolvido pela empresa Digilent.

3.1. Relação do módulos e organização do processador aritmético proposto

Dentre as arquiteturas relacionadas no item 2, e considerando as características e necessidades de um processador aritmético foi definida uma arquitetura a fluxo de dados com características que possibilitam a execução do mecanismo básico de execução de instruções a Fluxo de Dados descrito no item 2.4.1. A figura 15 mostra em diagrama de blocos a arquitetura proposta, composta por quatro módulos denominados: Unidade de Memória e Ativação, Unidade de Instruções Ativadas, Unidade de Operação e Unidade de Atualização de Dados. O texto que segue descreve as funções e características dos módulos.

A Unidade de Memória e Ativação utiliza uma memória de acesso aleatório (RAM) que armazena os códigos de operação, os valores (dados) a serem operados, o endereço de destino da operação e os marcadores da operação, dos operandos e do endereço. Um código de operação de soma, por exemplo, possui dois parâmetros, que são os valores a serem somados, então já é computado que para um endereço com o código de operação de soma, necessitará da validação de dois bits no campo dos marcadores para parametrizar os valores necessários. O armazenamento de instruções é descrito no item 3.1.1 que contém informações sobre a Unidade de Memória e Ativação.

O módulo denominado Unidade de Instruções Ativadas, descrito no item 3.1.2, que tem a função de enviar uma operação para a Unidade de Operação (item 3.1.3) assim que finalizada a execução da operação anterior, fator crucial para o bom funcionamento da arquitetura, pois também impede que haja ociosidade no processador.

A Unidade de Operação processa todas as informações através de algoritmos utilizados para a obtenção dos resultados esperados de cada uma delas. As informações são enviadas juntamente com seus parâmetros para serem executadas, tendo em vista que são disponibilizados quatro núcleos de processamento, visando permitir a execução de mais de uma operação ao mesmo tempo. Cada operação realizada é enviada para a Unidade de Atualização de Dados.

A Unidade de Atualização de Dados (descrita no item 3.1.4) é responsável por atualizar os endereços da unidade de memória para que utilizem o dado recém-processado. Neste módulo é fundamental que os endereços sejam atualizados de modo que possíveis informações já contidas no endereço de destino não sejam perdidas, pois há casos em que um operando, devido a dependência de dados, já estará contido no endereço, aguardando o resultado de uma outra operação para que seja disparada, desta forma o endereço precisa ser

atualizado com as novas informações. O diagrama de blocos da figura 15 mostra a arquitetura como um todo.

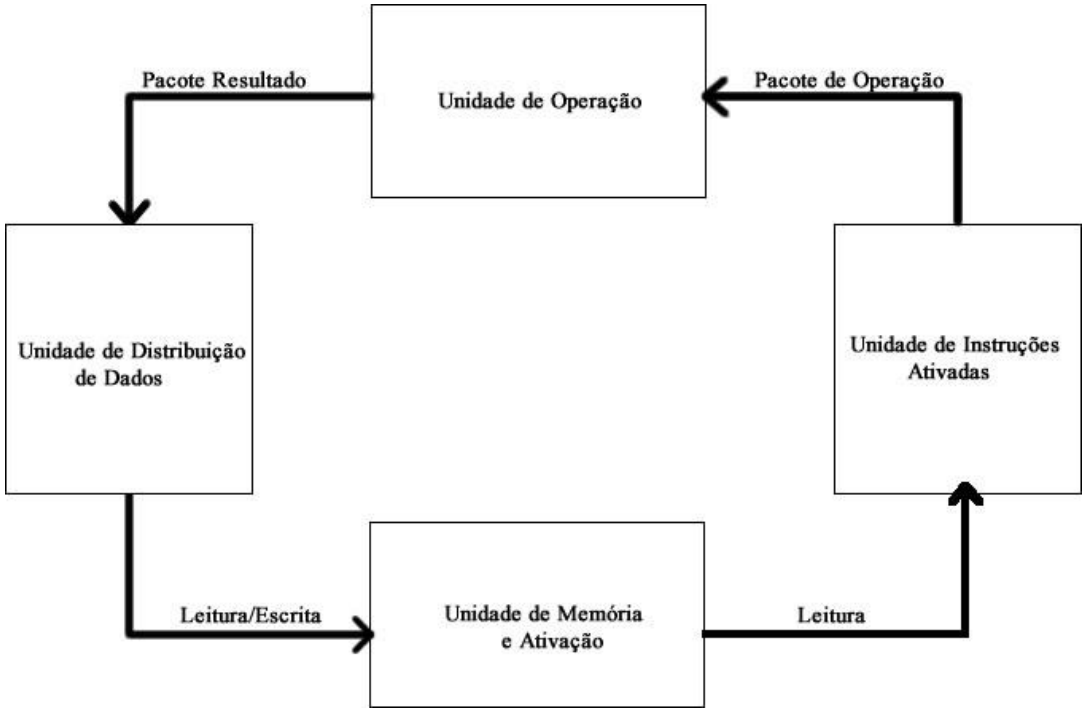

Figura 15: Diagrama de blocos da arquitetura proposta.

A arquitetura proposta é composta de quatro blocos principais, representados na figura 15. Os itens que seguem contêm a descrição de cada um dos módulos detalhadamente.

3.1.1. Unidade de memória e ativação

A unidade de memória e ativação é responsável pelo armazenamento dos programas de usuário de uma expressão matemática. Os códigos de operação e os valores a serem computados são armazenados para serem executados. O desenvolvimento deste módulo exigiu demasiado esforço para o desenvolvimento do projeto, onde o código gerado utilizando a linguagem de descrição de hardware VHDL conta com 1754 linhas necessárias para realizar todas as operações, validações e processos necessários para a obtenção do resultado esperado.

A Unidade de Memória é composta por uma memória de acesso aleatório (RAM). As operações e dados que são utilizados pela Unidade de Instruções Ativadas (item 3.1.2) vão, assim que disparadas, sendo adicionadas na fila de operações prontas para serem executadas.

A memória de acesso aleatório trabalha de forma que os marcadores informam o que deve ser feito. Os marcadores contêm 8 bits indicadores que armazenam informações importantes sobre a operação armazenada no endereço, ou seja, todos os endereços precisam, obrigatoriamente, ter preenchido o campo que os contém. São eles: Bit 7 - O fim da expressão como um todo, bit 6 - operando de destino na próxima operação (A ou B), bit 5 - quantidade de operandos no endereço atual, bit 4 - endereço livre ou reservado, bit 3 - operando A disponível, bit 2 - operando B disponível, bit 1 - endereço da próxima operação disponível, bit 0 - operador disponível. Os bits indicadores não necessários para que as operações e os seus dados sejam lidos e executados corretamente assim que estiverem disponíveis na memória (quando os bits do Operando A, Operando B, endereço da próxima operação e operação estiverem com o valor lógico "1").

A figura 16 a seguir mostra a memória principal (RAM) utilizada no projeto em tela, que pode ser implementada com a quantidade necessária de bits tanto nos operandos quanto na operação. Para utilizá-la como uma memória de acesso aleatório, é necessário apenas informar o endereço onde serão lidos ou escritos os valores. Para o uso do módulo, deve-se implementá-lo com a quantidade de bits de cada operando somado com a quantidade de bits da operação, além dos 8 bits dos marcadores.

A Unidade de Memória e Ativação deve possuir a mesma quantidade de núcleos operacionais que a Unidade de Operação, ou seja, no caso deste projeto possui quatro. Este fato se faz necessário devido ao fato de permitir a gravação e escrita simultânea em endereços distintos, onde é são realizadas diversas validações antes de um endereço ser acessado.

A tabela 2 também foi utilizada para exemplificar como o grafo a fluxo de dados contido na figura 5, que realiza a execução da expressão aritmética "$X = (A + B) * (A + C) * (B + C)$" ficaria alocado na Memória de Ativação.

Visando fins práticos a expressão aritmética citada conterá os seguintes valores: "$X = (3 + 1) * (3 + 2) * (1 + 2)$", sendo que o resultado "X", deve receber o valor inteiro e positivo "60". Vale ressaltar também que neste caso o programa será iniciado no endereço de número "50" para tornar o exemplo ainda mais prático. Tendo em vista que os operandos possuem uma marcação de identificação de positivo ou negativo no bit mais significativo, eles possuem 7 bits para armazenamento de valores, ou seja, um valor inteiro de no máximo 128 (0 a 127) na base 10. A operação possui 4 bits para o código de operação e 4 bits para identificação única (id) da operação realizada, ou seja, um valor inteiro de no máximo 256 (0 a 255). O endereço de destino

(*End. De Destino*) possui um valor inteiro entre 0 e 63 (64 endereços de destino possíveis), que é o tamanho da memória principal e os marcadores possuem 8 bits conforme descrito anteriormente neste item, onde cada bit possui uma representação.

A tabela estará representada no formato decimal para facilitar a compreensão da execução, porém os marcadores devem ser representados com valores booleanos (*0 ou 1*). Ao indicar o fim da expressão aritmética o endereço de destino não deve ser preenchido, pois o resultado ficará armazenado em um registrador visível para utilizar o valor onde for necessário.

Tabela 2: Expressão aritmética contida na figura 1 armazenada na memória principal (RAM).

End.	**Operando A**	**Operação**	**Operando B**	**End. Destino**	***Marcadores***							
					F	**Op.**	**Q**	**L**	**A**	**B**	**E**	**O**
0	X	X	X	X	X	X	X	X	X	X	X	X
1	X	X	X	X	X	X	X	X	X	X	X	X
...	X	X	X	X	X	X	X	X	X	X	X	X
50	3	+	1	53	0	0	1	0	1	1	1	1
51	3	+	2	53	0	1	1	0	1	1	1	1
52	1	+	2	54	0	1	1	0	1	1	1	1
53	R_{50}	*	R_{51}	54	0	0	1	0	0	0	1	1
54	R_{53}	*	R_{52}	NULL	1	0	1	0	0	0	1	1
55	X	X	X	X	X	X	X	X	X	X	X	X
...	X	X	X	X	X	X	X	X	X	X	X	X
63	X	X	X	X	X	X	X	X	X	X	X	X

A unidade de memória e ativação tem duas funcionalidades principais: receber instruções inserindo-as em endereços para poder executá-las (inserção de operações na memória, item 3.1.1.1) e verificar se todos os operandos estão disponíveis no endereço que for inserido ou alterado (verificação da disponibilidade de operandos e disparo de instruções, item 3.1.1.2), para o caso de dispará-las ou fazer com que fiquem aguardando até que todos os operandos estejam disponíveis.

3.1.1.1. Inserção de operações na memória

Para inserir uma informação na Unidade de Memória e ativação é necessário informar, primeiramente, quem está disponibilizando está operação, podendo ser a CPU ou o carregamento do conjunto de instruções para a memória.

A CPU informa apenas o resultado obtido da operação anterior, o operando e o endereço onde este valor será inserido, atualizando os marcadores do endereço de modo que fiquem atualizadas informando os dados que o endereço já possuía além dos dados que foram atualizados.

O carregamento do conjunto de instruções para a memória deve informar um conjunto maior de dados que a CPU, como o indicador se a operação inserida é a última a ser realizada pela expressão, a quantidade de operandos que a operação possui, o operando para o qual o valor irá no endereço de destino caso o resultado desta operação seja utilizado em outra operação e informar quais operandos serão disponibilizados inicialmente, podendo ser todos (neste caso será a operação disparada para execução), ou parcial (neste caso haverá o aguardo de uma resposta que atualize o operando restante neste endereço para que seja disparado).

Tanto a CPU quanto o carregamento do conjunto de instruções para a memória devem atualizar o bit 4 dos marcadores, que informa se o endereço atual está livre. Desta maneira, ambos podem ser o primeiro a utilizar o endereço, pois o outro apenas irá atualizá-lo com as informações restantes, evitando que haja um problema caso a CPU utilize o endereço antes mesmo do carregamento para a memória, pois a informação de utilização do endereço estará devidamente preenchida.

Vale ressaltar que a primeira validação realizada pela unidade de memória e ativação é a existência de algum endereço livre para inserção, caso contrário há uma fila de inserção para que armazene temporariamente as operações tanto da CPU quanto do carregamento das instruções.

3.1.1.2. Verificação da disponibilidade de operandos e disparo de instruções

Ao ser alterada ou inserida uma operação em algum dos endereços da memória, no próximo ciclo de relógio este endereço é verificado o estado da operação, sendo validado se todos os operandos, o operador e o endereço da operação de destino estão preenchidos com o

valor lógico “1”, o que representa a total disponibilidade de todos os operandos, e portanto, que a instrução será disparada. Vale ressaltar que algumas validações são realizadas durante este processo, são elas a utilização do endereço (se o mesmo se encontra livre, ou seja, bit 4 d os marcadores com o valor lógico “1”) e a quantidade de operandos, neste caso, se a operação possuir apenas um operando, é necessário apenas que este operando, o operador e o endereço de destino estejam disponíveis.

Para que os marcadores sejam manuseadas corretamente e disponibilizem o resultado esperado, a Unidade de Operação descrita no item 2.1.3, disponibiliza informações importantes sobre a execução das operações dentro de cada núcleo de processamento em tempo de execução. São elas a ordem de lotação nas filas de execução, informando o núcleo de processamento com a menor lotação na fila, o segundo, terceiro e o núcleo com a maior lotação, além do indicador informando os núcleos que estão executando alguma operação e os núcleos que não estão.

Utilizando estas informações a Unidade de Memória e Ativação (item 3.1.1), em cada núcleo operacional, também realiza algumas verificações antes do disparo da instrução, são elas a verificação da fila de execução com a menor quantidade de operações na espera na Unidade de Operação (item 2.1.3) juntamente com a verificação de prioridade de disparo.

A prioridade de disparo se faz necessária devido a possibilidade de duas operações estarem prontas para a execução ao mesmo tempo (no mesmo ciclo) em núcleos operacionais distintos da Unidade de Memória e Ativação neste caso, para não retardar o processo, o primeiro núcleo operacional possui a maior prioridade de execução na fila de execução com a menor quantidade de operações, o segundo núcleo operacional possui a segunda maior prioridade, ou seja, se o primeiro e o segundo núcleo operacional estiverem com a operação pronta para disparo, o primeiro núcleo envia à menor fila de execução e o segundo núcleo para a segunda menor fila, e assim consecutivamente com todos os núcleos disponíveis na arquitetura.

Em casos mais críticos como a lotação máxima de todas as filas de execução forem atingidas, as operações aguardam e a unidade de memória e ativação espera até alguma das filas sejam liberadas.

Para disparar uma operação pronta para execução quando as filas de execução da Unidade de Instruções ativadas (item 3.1.2) não estiverem com a lotação máxima atingida, a Unidade de Memória e Ativação remove a operação do endereço, insere na fila selecionada e libera o endereço.

3.1.2. Unidade de instruções ativadas

A Unidade de Instruções ativadas recebe um pacote de operação da Unidade de Memória e Ativação (item 3.1.1) e armazena em uma fila de instruções para cada núcleo de processamento da Unidade de Operação (item 3.1.3), onde elas serão executadas na ordem em que foram inseridas, porém o papel principal deste módulo é identificar quando a Unidade de Operação finaliza a execução de uma instrução para imediatamente enviar uma outra, impedindo que haja ociosidade no processamento. Esta identificação baseia-se na flag "*exec*" contida em cada um dos núcleos de processamento, que informa se o núcleo está executando alguma operação (*valor "1"*) ou não (*valor "0"*).

O desenvolvimento deste módulo fez necessária a geração de 126 linhas de código utilizando a linguagem de descrição de hardware VHDL visando desenvolver as validações necessárias para enviar as operações à Unidade de Operação sempre que possível.

3.1.3. Unidade de operação

Local onde todo o processamento é realizado. O desenvolvimento deste módulo foi realizado utilizando a linguagem de descrição de hardware VHDL e gerando um total de 850 linhas de código que contêm a implementação de todas as operações que a arquitetura é capaz de realizar, descritas neste tópico.

Esta unidade contém uma flag indicando se está ocupada ou livre para realizar o próximo cálculo, tornando possível o envio das operações apenas aos núcleos livres. Como as operações são fornecidas pela unidade de instruções ativadas (item 3.1.2), caso algum núcleo da unidade de operação esteja ocupado, a respectiva fila de operações prontas da unidade de instruções ativadas aguarda até que o núcleo de processamento a ela designado esteja livre, tendo em vista que cada fila envia instruções a apenas um núcleo de processamento.

Para que haja o processamento paralelo, deve haver mais de um núcleo de processamento (Unidade Aritmética - local onde os cálculos são realizados), porém em nível de desenvolvimento em FPGA, será utilizada a quantidade de núcleos que for possível dentro do mesmo, pois o software de desenvolvimento do circuito não reage de maneira positiva em versões de 64 bits, porém a versão de 32 bits gera overflow na memória ao tentar compilar o

projeto com mais de quatro núcleos de processamento e com a memória de vídeo que será descrita nos itens 3.2 e 3.3.

Um estudo de caso de foi realizado para que seja possível medir a quantidade máxima de núcleos de processamento que foi utilizada sem que exceda os limites de compilação do software ou da placa, e devido a arquitetura como um todo e as unidades desenvolvidas para testes mais aprofundados, a quantidade de Unidades Aritméticas foi definida em quatro, que permite um alto grau de paralelismo e é completamente suportado pelo programa de desenvolvimento de 32 bits.

Vale ressaltar que a Unidade de Operação é capaz de executar instruções iônicas, onde é possível escolher quais bits serão utilizados em cada operação, sendo que as operações que poderão ser realizadas são: Soma (*0000 - soma dois valores passados como parâmetro*), subtração *(0001 - subtrai dois valores passados como parâmetro*), multiplicação (*0010 - multiplica dois valores passados como parâmetro*), divisão (*0011 - divide dois valores passados como parâmetro*), resto (*0100 - pega o resto da divisão de dois valores passados como parâmetro*), deslocamento para a direita - multiplicar por 2 (*0101 - multiplica um valor passado como parâmetro por dois elevado pela quantidade passada como um segundo parâmetro*), deslocamento para a esquerda - dividir por 2 (*0110 - divide um valor passado como parâmetro por dois elevado pela quantidade passada como um segundo parâmetro*), not (*0111 - o inverso de um bit ou o inverso de um número com mais de um bit – positivo ou negativo*), and (*1000 - operação lógica de um bit "E"*), or (*1001 - operação lógica de um bit "OU"*), xor (*1010 - operação lógica de um bit "XOU")*, nor (*1011 - operação lógica de um bit "NOU"*) e cmp (*1100 - operação relacional entre dois valores passados como parâmetro, informando igualdade, diferenciação, maioridade, menoridade, maioridade ou igualdade e menoridade ou igualdade*).

Devido ao fato de nos dias atuais a quantidade de núcleos de processamento ser sempre menor do que a quantidade de operações a serem executadas, foi criada uma fila de instruções prontas para a execução para cada núcleo de processamento, permitindo que sejam operações sejam adicionadas às filas sempre que o núcleo escolhido estiver executando a uma outra e evitando a perda de operações. A criação das filas também permite que haja um balanceamento homogêneo de carga, tendo em vista que a Unidade de Operação informa em suas portas de saída a quantidade de operações pendentes cada fila possui já identificando a ordem de melhor escolha dos núcleos para o envio de operações, onde o melhor núcleo para ser escolhido é aquele que possui a menor quantidade de operações pendentes na fila.

Operações aritméticas, lógicas e relacionais foram implementadas em todos os núcleos de processamento, que são idênticos uns aos outros para que seja possível uma seleção otimizada de núcleo de execução. Visando eliminar a ociosidade do processador como um todo, as filas de instruções prontas para a execução de cada núcleo de processamento enviam uma operação assim que a flag "*exec*" assumir o valor lógico "*0*", indicando que a operação que estava em execução foi finalizada e já é possível executar a uma outra, uma vez que o ponto principal do projeto é eliminar a ociosidade de todos os núcleos tanto quanto for possível.

Vale ressaltar que cada núcleo de processamento será uma Unidade Aritmética funcionando em paralelo com as outras, permitindo a execução de diversas operações simultâneas, ou seja, aumentando o paralelismo e, consequentemente, a velocidade de execução.

A figura 16 mostra a Unidade de Operação com uma Unidade Aritmética nela contida porém o processador desenvolvido possui quatro unidades aritméticas, onde cada uma delas é capaz de executar um cálculo simultaneamente às outras. A *flag* "*exec*" é responsável por indicar se a respectiva Unidade Aritmética está executando uma operação. Uma vez que iniciada uma operação, a Unidade Aritmética não para sua execução ou troca de operação enquanto esta não é finalizada.

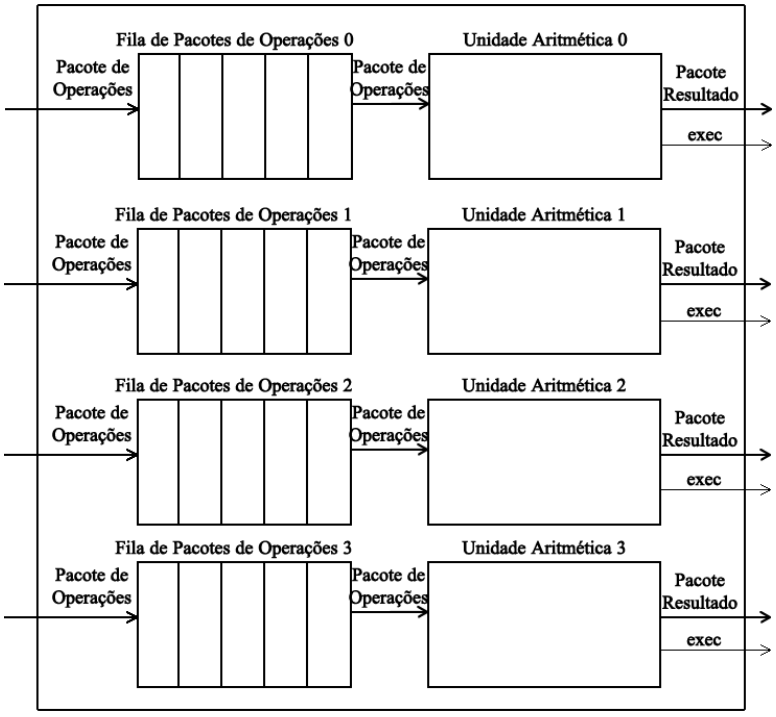

Figura 16: Unidade de Operação contendo as Unidades Aritméticas.

Um aspecto importante da Unidade de Operação é a diferença entre o formato de instrução anterior e posterior a execução, onde o pacote resultado da operação realizada possui o formato de atualização na unidade de memória e ativação, ou seja, possui apenas o operando de saída, o endereço de destino e os marcadores, onde o endereço de destino indica qual posição da memória o operando será inserido, os marcadores indicam em qual operando do endereço de destino o resultado será colocado e se o resultado desta operação é o resultado final da expressão como um todo, em caso positivo, o valor é enviado para um registrador onde o resultado possa ser utilizado, em caso negativo o valor é enviado para a unidade de atualização de dados, que será descrita no item 2.1.3.

A tabela 3 que segue mostra o formato de instrução do pacote resultado gerado pela Unidade de Operação após ser processado. Os valores expressados nesta tabela representarão os pacotes resultado gerados pela expressão aritmética "$X = (3 + 1) * (3 + 2) * (1 + 2)$" exibida na tabela 2.

Os resultados gerados a partir das operações podem ser utilizados em outras operações, portanto inseridos em outro endereço da memória principal permitindo sua utilização. Com isso a continuidade na execução da expressão aritmética como um todo, de modo que o resultado final obtido seja o correto de acordo com as normas matemáticas de execução de expressões linearizadas.

Tabela 3: Representação dos pacotes resultado gerados pela expressão contida na tabela 2.

Resultado	**End. de Destino**	***Marcadores***							
		F	**Op.**	**Q**	**L**	**A**	**B**	**E**	**O**
4	53	0	0	0	0	0	0	0	0
5	53	0	1	0	0	0	0	0	0
3	54	0	1	0	0	0	0	0	0
20	55	0	0	0	0	0	0	0	0
60	NULL	1	0	0	0	0	0	0	0

Vale ressaltar que no pacote resultado, os únicos marcadores utilizados são o do bit 7, que representa o fim da expressão aritmética e o do bit 6, que representa em qual operando do

endereço de destino o resultado será inserido. Os demais valores não são utilizados porém devido ao padrão dos marcadores eles são mantidos com a finalidade de manter o mesmo formato em toda a arquitetura.

3.1.4. Unidade de atualização de dados

Este módulo, que contém um total de 136 linhas de código VHDL gerado, foi desenvolvido com a função de atualizar os resultados obtidos nos endereços e operandos corretos da Unidade de Memória e Ativação (item 3.1.1). É dever da Unidade de Atualização de dados não só inserir o valor no local correto, mas também atualizar os marcadores da Unidade de Memória e Ativação de modo que não haja alteração no resultado final da expressão.

Para realizar a atualização da forma como deve, este módulo realiza algumas validações, como a verificação se o endereço de destino está livre. Em caso positivo o valor é inserido no operando que informado nos marcadores do pacote resposta da Unidade de Operação (item 3.1.3) e o endereço é atualizado para ocupado. Em caso negativo o endereço de destino é atualizado com o valor de resposta sem que as informações já contidas no endereço sejam perdidas, pois este resultado fará parte de uma operação a ser executada que necessita de todos os operandos disponíveis para ser disparada, ou seja, se o endereço de destino não estiver livre a unidade de atualização irá inserir o operando onde é necessário e atualizar os marcadores do endereço de destino de modo a informar que apenas mais este operando está disponível não importa qual deles seja, sem afetar os valores previamente inseridos pelo carregamento de instruções ou pela Unidade de Operação.

3.2. Síntese, implementação e validação do processador aritmético

Este item destina-se a especificar o processo de síntese, implementação e validação do processador aritmético desenvolvido no projeto, tais como os módulos complementares desenvolvidos para a finalidade de testes, dificuldades encontradas durante o desenvolvimento, soluções utilizadas em cima das dificuldades e a etapa de testes para validar o que foi construído.

3.2.1. Dificuldades e soluções do projeto

A arquitetura a fluxo de dados desenvolvida contém quatro módulos principais. A Unidade de Memória e Ativação (item 3.1.1) armazena os pacotes de operação para dispará-los quando estiverem com todos os operandos disponíveis para a Unidade de Instruções Ativadas (item 3.1.2), que detecta o melhor núcleo da Unidade de Operação (item 3.1.3) para o qual cada operação deve ser enviada naquele instante. A Unidade de Operação processa todas as requisições e envia os resultados para a Unidade de Atualização de Dados (item 3.1.4) que é responsável por fazer com que o resultado de uma operação seja inserido novamente na memória para utilizá-lo em outra operação ou disponibilizá-lo em um registrador da arquitetura caso seja a última operação da expressão aritmética para poder utilizá-lo para qualquer outra finalidade.

Para realizar sua tarefa corretamente, a Unidade de Atualização de Dados verifica se o resultado obtido é o resultado final da expressão aritmética como um todo ou se é apenas o resultado de uma das operações da expressão, então decide se o resultado é enviado a um registrador ou à memória para continuar a execução através do bit 7 dos marcadores que ficam armazenadas na Unidade de Memória e são transferidos pela Unidade de Operação da maneira que chegaram até ela.

Para solucionar a ociosidade também foram gerados indicadores de quantidade de ocupação na fila de execução de cada núcleo da unidade de operação, informação essa que é apresentada para a Unidade de Memória e Ativação de modo que são informadas a menor fila de execução, a segunda menor fila de execução, a terceira menor fila de execução e a maior fila de execução. Baseado nessas informações os núcleos da memória principal verificam para qual núcleo da unidade de operação uma operação será enviada.

A Unidade de Memória e ativação possui quatro entradas e saídas de dados, assim como a quantidade de núcleos do processador. Uma operação pode entrar na Unidade de Memória e Ativação pelo núcleo número 2, por exemplo, e sair pelo núcleo número 1, se a fila de execução número 1 estiver com a menor lotação e o núcleo 1 não estiver tentando inserir uma operação na mesma fila de execução que o núcleo 2 ao mesmo tempo.

Para evitar que mais de uma operação tente acessar a mesma fila de execução no mesmo ciclo foram criadas prioridades dos núcleos da memória principal, onde o primeiro núcleo tem a prioridade de acessar a menor fila de execução em relação aos outros, o segundo

núcleo possui prioridade em relação aos outros dois e o terceiro núcleo possui prioridade em relação ao último. Para que este fato tenha se tornado possível, ao disparar uma operação, cada núcleo da memória verifica se algum núcleo anterior já também está pronto para disparar alguma operação, e então escolhe o núcleo de disparo com a menor fila de execução possível de acordo com os núcleos de memória com maior prioridade.

Uma questão a ser levada em consideração é a falta da necessidade de estar constantemente verificando a memória principal, fato que economiza tempo de busca. Assim que uma instrução é inserida ou alterada em algum endereço, no passo seguinte a arquitetura verifica se o endereço está pronto para ser executado, caso contrário ele permanece na espera até que seja modificado novamente. Ao alterar o endereço uma nova validação é realizada verificando se a operação pode ser disparada. Ao ser disparada a operação, o endereço é liberado automaticamente para o armazenamento de outras.

Um fator determinante é o indicador de endereços livres, que é atualizado sempre que um endereço é ocupado ou liberado, informando sempre o menor endereço livre disponível para uso. Quando a Unidade de Memória e Ativação está em busca de endereços livres uma marcação é ativada até que seja encontrado o primeiro endereço livre ou o erro de overflow seja exibido. Ao encontrar o primeiro endereço livre a arquitetura volta a inserir operações na Unidade de Memória e Ativação, porém a busca de mais endereços livres continua, até que se encontre o próximo, que fica armazenado em um registrador, acessado ao tentar inserir outra operação, ou informe o erro de overflow, então outra busca é inicializada sem paralisar a arquitetura, ou seja, a arquitetura só é paralisada se não houver mais endereços livres.

3.2.2. Módulos complementares necessários desenvolvidos

Durante o desenrolar do projeto foi necessário o desenvolvimento de alguns módulos complementares para a finalidade de testes da arquitetura proposta tais como o módulo de conexão com a saída de vídeo do tipo VGA (item 3.2.2.1) e o compilador simplificado para a arquitetura proposta (item 3.2.2.2), que agora serão descritos. Essas atividades complementares exigiram esforços e tempo que possibilitaram realizar a depuração e validação de todo o projeto.

3.2.2.1. Módulo de conexão com a saída de vídeo do tipo VGA

A construção deste módulo foi realizada para possibilitar e viabilizar a depuração e validação do processador aritmético com arquitetura a fluxo de dados construído, onde no monitor são exibidos quatro quadrados onde cada um representa um núcleo de processamento, podendo eles assumir a cor verde quando estiverem executando alguma operação ou a cor vermelha quando não estiverem, indicando que há ociosidade

O desenvolvimento do módulo de conexão com a saída VGA visando a visualização do estado de execução dos núcleos de processamento no monitor utilizou 126 linhas de código escrito na linguagem de descrição de hardware VHDL.

Para construir a conexão com a saída VGA foi necessário que, primeiramente, uma memória de vídeo utilizada para representar os pixels em uma saída VGA fosse construída, onde a conexão e sincronização com o vídeo também necessitou de implementação utilizando esta memória.

O sincronismo com a saída de vídeo necessita de um relógio que contém a velocidade de oscilação de 25MHz, ou seja, 25% da capacidade total do FPGA utilizado. Após reduzir a frequência de oscilação do relógio que atua neste sincronismo para a velocidade necessária é necessário que sejam percorridos todos os pixels da tela, pois eles são visualizados individualmente no monitor utilizando a frequência de 60Hz, ou seja, todos os pixels da tela são percorridos 60 vezes em um segundo, fato que impede que olhos humanos percebam que os pixels são exibidos individualmente, fornecendo a sensação de visualizar todos simultaneamente.

Para a exibição das informações no monitor, a memória de vídeo necessitou possuir tamanho suficiente para suportar 800 X 600 pixels, um total de 480000 pixels, necessitando de 512 KBytes (512000 bytes) de endereços de 8 bits cada (tamanho da palavra utilizada pelo padrão VGA).

Para percorrer todos os pixels da tela, utilizando a frequência de 25MHz deve-se percorrer 480000 endereços da memória de vídeo (resolução utilizada), um a cada oscilação do relógio, fazendo com que no momento que um pixel for exibido seja fornecido a cor armazenada em seu endereço da memória.

As informações para testes foram exibidas no monitor utilizando a saída VGA, tendo em vista que o módulo responsável por esta função foi implementado, testado e validado.

O código VHDL gerado consiste em percorrer todos os pixels da memória de vídeo relacionando-o com uma linha e coluna de pixels do monitor a cada ativação do oscilador de 25MHz, e utilizar o valor inserido em cada endereço para exibir na saída VGA do componente, onde este pixel (pertencente a determinada linha e coluna) será exibido pelo monitor de acordo com o valor da cor armazenada, estando este valor no padrão RGB, ou seja, o monitor exibe um pixel por vez onde a tela é atualizada em uma velocidade de 60Hz.

3.2.2.2. Compilador simplificado para a arquitetura proposta

Foi também implementado um compilador simplificado que realiza a transformação de uma expressão matemática escrita de forma linearizada em um grafo a fluxo de dados. O respectivo grafo a fluxo de dados de uma expressão matemática é inserido na memória de instruções do processador aritmético. Atualmente o compilador é capaz de organizar a expressão linearizada em forma de árvore, onde a separação é realizada por escopo, ou seja, abertura e fechamento de parênteses, e por importância, onde operações mais importantes como multiplicação e divisão devem estar em níveis mais elevados da árvore quando estiverem no mesmo escopo. Para gerar a árvore de precedência foi dividia a expressão linearizada em um vetor de Strings para que seja dividido um nó em filhos até que cada filha da árvore possua apenas uma String em seu vetor, podendo assim executar as expressões do maior nível para o menor nível da árvore e da esquerda para a direita, conforme descreve a figura 17 que segue.

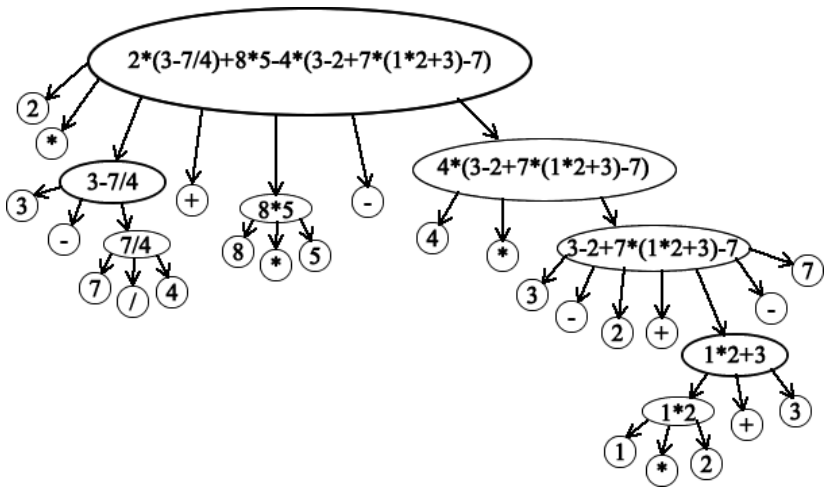

Figura 17: Árvore gerada a partir de uma expressão aritmética.

A árvore implementada é totalmente recursiva, onde é inserido um vetor de Strings em sua raiz e cada elemento do vetor é um token. Um token é cada valor da expressão aritmética, como um operador ou um operando. A árvore deve expandir-se automaticamente até que todas as suas folhas contenham um token, sendo uma folha um nó que não possui nós derivados dele, ou seja, em um nó folha o vetor de tokens possui apenas um token em seu valor. Vale ressaltar que um nó da árvore é um vetor de tokens que deve ser analisado e expandido se necessário.

O ato recursivo deve ser disparado ao definir o valor da raiz, sendo a raiz o primeiro nó da árvore, ou seja, o nó com o menor nível da árvore, o nível zero. A partir de então a precedência de escopos e importância de operações deve criar outros nós conforme o necessário. O escopo é identificado pela abertura e fechamento de parênteses, onde a cada abertura de parênteses o escopo é incrementado, portanto a cada fechamento de parênteses o escopo é decrementado.

Para detectar a dependência de dados entre as operações ao gerar os endereços da memória onde serão alocadas, deve-se primeiramente definir um endereço da memória principal distinto para cada nó que contém uma operação a ser realizada, a partir de então basta unificar os nós que são folhas em operações organizando-os do maior nível para o menor (raiz) e da esquerda para a direita para gerar os endereços partindo das operações mais importantes para as operações menos importantes, sendo o endereço de destino o mesmo que o do nó que o gerou executando-se a raiz, que não possui endereço de destino. A raiz preencherá o marcador "*Fim*" do campo "*Marcadores*" da memória principal com o valor lógico "*1*", indicando o fim da expressão como um todo e fazendo com que o resultado seja disponibilizado em um registrador para ser utilizado onde for necessário.

A estrutura da árvore permite que o resultado ao finalizar a execução da expressão aritmética seja o desejado pois faz com que a precedência de operações seja obedecida, e portanto as regras matemáticas.

Ressalta-se ainda que para inicializar o processo de expansão da árvore é necessário que a expressão linearizada esteja validada quanto a caracteres permitidos e quantidade de abertura e fechamento de parênteses, por exemplo.

O tempo disponível para implementação do projeto inviabilizou tarefa de compilação de expressões matemáticas para o processador desenvolvido, onde a parte de geração de

endereços para cada operação no formato de instrução da memória principal não pôde ser desenvolvida. Assim a tarefa será realizada em fases posteriores do projeto.

3.2.3. Ambiente de desenvolvimento

As atividades desenvolvidas contribuíram para realizar a síntese e implementação de todos os módulos da arquitetura, que integram o processador aritmético com arquitetura a fluxo de dados proposta. Para descrever o circuito projetado foi utilizada a linguagem de descrição de hardware VHDL juntamente com o software ISE Project Navigator v0.86 para editar o código-fonte.

3.2.3.1. Ambiente de desenvolvimento e testes de hardware

Para a construção dos testes necessários durante o desenvolvimento do projeto o circuito foi inserido em componente FPGA utilizando o software ISE Project Navigator v0.86 para simulações utilizando a saída VGA (item 3.2.2.1) em um loop infinito de operações dependentes. Vale ressaltar que componente FPGA utilizado, é denominado Nexys-3™ Spartan-6 pela Digilent (fabricante da placa) conforme exibe a figura 18 que segue.

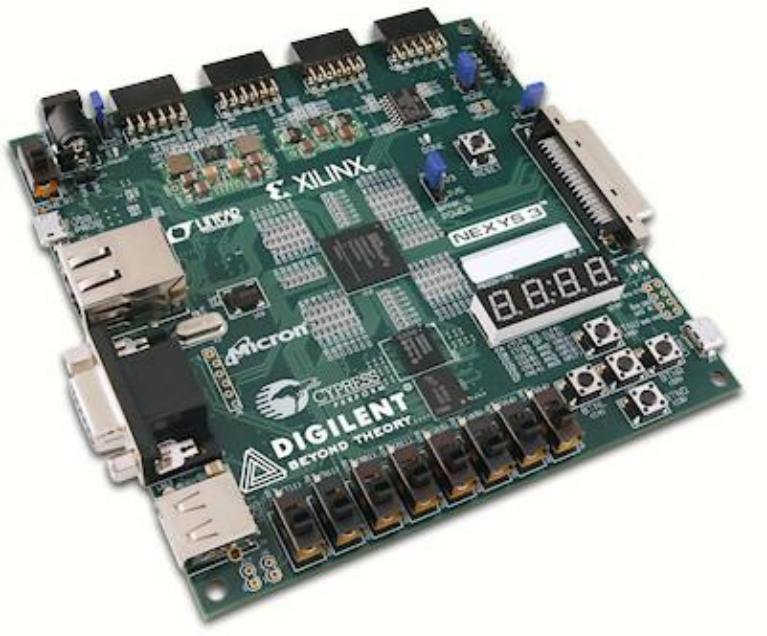

Figura 18: FPGA modelo Nexys 3™ Spartan-6 fabricado pela Digilent utilizado no projeto.

Para simulações visualizando todos os passos realizados pela arquitetura foi utilizado o software ModelSim. O circuito projetado foi testado e validado pelos dois tipos de simulação citados neste item com a finalidade de verificar o circuito minuciosamente desde o resultado de uma simples operação até o resultado de um conjunto relevante de operações paralelas.

Para gerar o loop infinito de operações no FPGA foi desenvolvida uma iteração com o componente onde é possível inserir operações em endereços da memória utilizando os switches do componente e seus botões, visualizando o endereço que está sendo modificado através do display de 7 segmentos. A implementação deste módulo demandou tempo de desenvolvimento necessário para validar a arquitetura de forma eficiente.

Após inserir todas as operações, com o monitor utilizado conectado com a saída VGA está exibindo todos os quadrados na cor vermelha devido ao fato de não haver operações em execução. O botão mais à direita ao lado dos switches faz com que todas as operações inseridas sejam disparadas quando desejado, repetindo-as em um loop infinito sempre que a execução de todas elas são finalizadas, mantendo o processador executando operações até que o botão mais à esquerda ao lado dos switches seja apertado.

O processo de testes demandou muito tempo para que fosse possível inserir operações dependentes entre si para serem executadas, porém o resultado obtido está dentro do esperado pois uma vez que disparadas as operações, os quadrados que representam os núcleos de processamento não são exibidos na cor vermelha para os olhos humanos até que a execução das operações seja interrompida, fato que demonstra que há ociosidade mínima.

Para confirmar tal informação os testes realizados no software ModelSim, que exibem o estado de todos os sinais da máquina a cada passo realizado, demonstram que a arquitetura utiliza um intervalo de até cinco passos entre uma operação e outra até que estas sejam finalizadas, pois no pior caso, quando uma operação está aguardando outra em execução e resta menos operações do que núcleos de processamento para serem executadas.

Assim que finalizada a execução da operação atual são necessários cinco passos desde sua atualização na memória principal até o processamento da próxima, caso contrário haverão operações bem distribuídas nas filas de execução prontas para inicializarem assim que finalizadas as operações já atualmente em execução.

3.3. Dimensionamento e Estudo (Processador X Field Programmable Gate Array – FPGA X Ambiente de desenvolvimento)

Após a validação de todas as unidades implementadas, foi realizado o dimensionamento da quantidade de Unidades Aritméticas que compõe o processador aritmético com arquitetura a fluxo de dados em tela em relação ao FPGA e ao software de desenvolvimento da arquitetura.

Um Field Programmable Gate Array, conhecido como FPGA é um módulo desenvolvido para fins de testes práticos quando se trata de criação de circuitos eletrônicos. Cada modelo de FPGA contém uma quantidade limitada de recursos, o que significa que cada projeto necessita um nível de sofisticação do componente, porém este fato afeta diretamente o custo da placa a ser adquirida.

Internamente o FPGA é capaz de simular o circuito desenvolvido e exibir os resultados através de suas saídas, onde os sinais e conexões destas saídas também devem ser programados de modo a exibir corretamente o valor do cálculo principal realizado pelo circuito.

Quando o componente é ligado ele não é capaz de executar processos ou rotinas, pois não há programação prévia armazenada, então a partir de softwares desenvolvidos pela Digilent em parceria com a Xilinx, a placa é capaz de se comunicar com computadores pessoais através da entrada USB (Universal Serial Bus), onde é possível compilar o projeto e gerar um arquivo que a placa consegue compreender e simular, representando o que foi programado no projeto em um circuito que é capaz de interagir com o mundo externo através das entradas saídas nele existentes.

Para desenvolver o projeto, devido ao custo destas placas ser elevado, a placa adquirida foi o modelo Nexys 3^{TM} Spartan-6, que é uma placa com a qual o projeto pôde ser desenvolvido. A tabela 4 que segue exibe o nível de utilização do componente com o projeto comparado com o suportado pela placa. As demais partes lógicas do FPGA não citadas possuem a porcentagem utilizada pelo projeto em 0%. A tabela 5 exibe o tempo de execução das instruções no pior e no melhor caso.

Tabela 4: Porcentagem de utilização do FPGA.

Parte Lógica	**Quantidade Total**	**Porcentagem Utilizada**
Número de Registradores	18224	1%
Número de “Look up Tables”	9112	1%
Fatias Utilizadas	2278	1%
Multiplexes Utilizados	4556	1%
Relógios globais multiplexados	232	6%
Memória RAM Utilizada	2Mb	28,125%
Pinos Físicos Utilizados	232	4%
Pinos de Entrada e Saída alocados utilizados	11	100%
Velocidade de Relógio	100MHz	98%

Tabela 5: Tempo de execução das operações.

Operação	**Pior caso**	**Melhor caso**
Soma	1 passo	1 passo
Subtração	1 passo	1 passo
Multiplicação	Quantidade de passos igual a de bits	1 passo
Divisão	Quantidade de passos igual a de bits * 1,5	1 passo
Resto	Quantidade de passos igual a de bits * 1,5	1 passo
Deslocamento à Direita	1 passo	1 passo
Deslocamento à Esquerda	1 passo	1 passo
Operações lógicas	1 passo	1 passo
Operações Relacionais	1 passo	1 passo

Foi necessária a realização de um dimensionamento da Unidade de Operação em comparação com o FPGA e com o ambiente de desenvolvimento (item 3.2.3), pois o projeto está baseado na quantidade de núcleos de processamento que haverá neste módulo. Assim que implementado um núcleo de processamento da Unidade de Operação com todas as operações necessárias, foram realizadas diversas simulações para verificar a quantidade de recursos utilizados do FPGA e do ambiente de desenvolvimento no momento da compilação com apenas uma cópia do núcleo simulado. Esse processo possibilitou realizar o dimensionamento de todos

os módulos da arquitetura proposta para viabilizar a construção do processador aritmético utilizando o componente FPGA disponível e os softwares Project Navigator e ModelSim citados no item 3.2.3.

Devido a limitação do software de desenvolvimento com a versão de 32 bits no momento da compilação gerando estouro da memória (overflow), foram necessárias a limitação da arquitetura em tamanho da palavra para 8 bits e da quantidade de endereços da memória principal para 64 endereços de quatro palavras de 8 bits cada, somando-se 32 bits para as palavras dos 2 operandos, a palavra do operador, de 8 bits (4 para a operação a ser realizada e 4 para o identificador único da operação - ID), e da palavra dos marcadores (utilizada na validação de utilização das operações), também de 8 bits. Além das dimensões da memória principal, para respeitar os limites de compilação citados, ainda foram utilizados quatro núcleos de processamento na arquitetura, ressaltando que a memória de vídeo também utiliza muito deste processo.

A quantidade de núcleos de processamento da arquitetura, apesar de limitada a quatro, não prejudicou os testes de paralelismo de dependência de dados.

O processador proposto pode ser implementado utilizando-se o tamanho que for necessário respeitando os limites do ambiente de desenvolvimento tanto em tamanho da palavra e quantidade de núcleos de processamento quanto em quantidade de palavras possíveis na memória principal, tendo em vista que os marcadores devem ser preenchidos corretamente para que a expressão matemática retorne o resultado esperado.

4. Conclusões e propostas de continuidade

Após a sintetização e validação do processador proposto, o resultado encontrado foi uma maneira eficiente de lidar com a dependência de dados, pois não há espera de disparo de operações quando estão prontas para serem executadas devido ao fato do disparo imediato de uma instrução quando seus operandos estiverem disponíveis, conforme implementado e validado na unidade de memória e ativação (item 3.1.1).

A quantidade de núcleos de processamento disponíveis é a única restrição para aumentar ou diminuir a velocidade de execução das expressões matemáticas, pois uma operação pronta para execução é imediatamente disparada.

Vale ressaltar que processadores que utilizam arquiteturas a Fluxo de Dados possuem uma ótima solução para controle de processos (Bugatti, Ildeberto de Genova, 1986), como processos de comutação para gerar tomadas de decisão, onde o circuito deve decidir o que fazer baseado nos dados informados, pois sempre que todas as informações são fornecidas a decisão é imediatamente processada. Outra maneira de utilizar fluxo de dados é aplicar em escopos específicos, como o utilizado neste projeto para aplicações matemáticas, permitindo que a definição automática da ordem de precedência seja utilizada naturalmente.

Para a continuidade do projeto proposto tem-se a adaptação do processador para operações de ponto flutuante e ao uso geral. A partir de então o foco será em aperfeiçoar um compilador para a arquitetura de modo que seja possível programar instruções complexas com o circuito projetado.

O projeto também pode ser continuado na tentativa de utilizar processadores a Fluxo de Dados em máquinas de uso geral utilizando a estrutura de uso geral de modo que operações necessárias sejam enviadas a um módulo de processamento de operações lógicas e matemáticas a fluxo de dados, onde o processador de uso geral apenas submete as operações e aguarda os resultados.

Os passos seguintes são desafiadores e demandam pesquisas e dedicação, porém representa a amplitude que ainda é possível obter com base no projeto desenvolvido, o que significa que é possível que processadores melhorem o desempenho de máquinas com arquiteturas convencionais através do uso processadores aritméticos com arquitetura a Fluxo de Dados utilizando as características propostas neste projeto.

5. Referências Bibliográficas

ACKERMAN, W. B.. *data flow* **Languages**. IEEE Computer, Fevereiro, 1982. p. 15-25.[3]

AGERWALA, T. & ARVIND.. *data flow* **Systems**.IEEE Computer,Fevereiro,1982. p.10-4.

BUGATTI, ILDEBERTO G.. **Computadores a Fluxo de Dados: Aplicação em Central de Comutação Telefônica**. 1986. 115 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual de Campinas, Campinas, 1986. p.11-37.

BUGATTI, I. G. & MOTOYAMA, S.. **Computadores Baseados em Fluxo de Dados: Conceitos e Aplicações**. DEE.FEC.UNICAMP., RI04, 33p., Julho, 1986.

BUGATTI, I. G. & MOTOYAMA, S.. **Central de Comutação controlada por um Computador a Fluxo de Dados**. Quarto Simpósio Brasileiro de Telecomunicações (SBT). Março, 1986. p.115-20.

BUGATTI, I. G. & MOTOYAMA, S.. **Uma Proposta de Arquitetura Fluxo de Dados para Controlar uma Central de Comutação**. DEE.FEC.UNICAMP., RI05,32p., Julho, 1986.

DENNIS, J. B.. & MISUMAS, D. P.. **A Preliminary Architecture for a Basic *data flow* Processor**. In: Annual Symp. Computer Architectures, 2. Proceedings. 1975. p.126-32.

GURD, J. R.; KIRKHAN, C. C.; WATSON, I.. **The Manchester Prototype *data flow* Computer**. Communication of the ACM, 28, (1): 34-52, 1985.

RUMBAUGH, J.. **A *data flow* Multiprocessor**. IEEE Translactions on Computer, Fevereiro, 1977. p.138-46.

6. Anexos

6.1. Documentação do componente FPGA utilizado

Para ter acesso a documentação fornecida pela Digilent sobre o FPGA utilizado, deve-se acessar o link que segue referente a página oficial do FPGA na web fornecida pelo fabricante: http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3&CFID=6628344&CFTOKEN=eb0d9c1fe9c5a5e2-4200A1A7-5056-0201-02E22D2488ACDEC5

6.2. Legenda das tabelas contidas no projeto

End.: Endereço;

X: Representa qualquer valor, pois não afeta o resultado da operação contida entre o endereço 50 e 55 para fim de exemplo prático;

R_n: Representação simbólica do resultado da operação realizada no endereço identificado pelo número que deve substituir o "n". Por exemplo "R50", indica que está esperando o resultado do endereço 50. Vale ressaltar que a arquitetura não identifica o paralelismo baseado nesta representação, e sim no endereço de destino e nos marcadores;

F: Representa o fim da expressão aritmética como um todo ("*0" = Não ou "1" = Sim*);

Op.: Representa o operando de destino no endereço de destino (*A = "0" ou B = "1"*);

Q: Representa a quantidade de operandos que a operação do endereço atual possui ("*0" = 1 operando ou "1" = 2 operandos*);

L: Indica se o endereço atual está sendo utilizado (*valor "0"*) ou se está livre (*valor "1"*);

A: Indica se o Operando A está disponível para execução (*valor "1"*) ou não (*valor "0"*);

B: Indica se o Operando B está disponível para execução (*valor "1"*) ou não (*valor "0"*);

E: Indica se o Endereço de Destino está disponível para execução (*valor "1"*) ou não (*valor "0"*);

O: Indica se o Operador está disponível para execução (*valor "1"*) ou não (*valor "0"*);

NULL: Indica que o valor não será utilizado.