Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Um Ambiente Computacional para apoio ao Ensino de Estrutura de Processamento

Adriane Bellé

Orientador: José Raimundo de Oliveira

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Banca Examinadora:

Prof. Dr. José Raimundo de Oliveira (Presidente)

Prof. Dr. Mauricio Araujo Dias (FCT/UNESP)

Prof. Dr. Furio Damiani (FEEC/UNICAMP)

Campinas - SP
Dezembro de 2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

	Bellé, Adriane
B415a	Um ambiente computacional para apoio ao ensino de estrutura de processamento / Adriane Bellé. --Campinas, SP: [s.n.], 2011.
	Orientador: José Raimundo de Oliveira.
	Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.
	1. Engenharia de computador. 2. Simulação (Computadores). 3. Emuladores (Programas de computador). 4. Microprocessadores (Projetos e construção). 5. Computadores (Estudo e ensino). I. Oliveira, José Raimundo de. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: A computational environment to support the teaching of structure processing
Palavras-chave em Inglês: Computer engineering, Computer (Simulation), Emulators (Design abd construction), Computers (Estudy and teaching)
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Maurício Araujo Dias, Furio Damiani
Data da defesa: 22-12-2011
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA – TESE DE MESTRADO

Candidata: Adriane Bellé

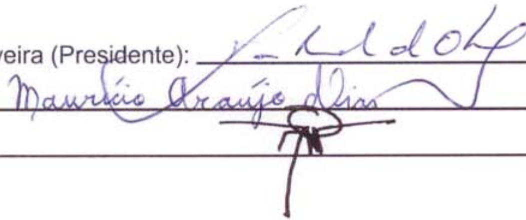
Data da Defesa: 22 de dezembro de 2011

Título da Tese: "Um Ambiente Computacional para apoio ao Ensino de Estrutura de Processamento"

Prof. Dr. José Raimundo de Oliveira (Presidente): ______________________

Prof. Dr. Mauricio Araujo Dias: ______________________

Prof. Dr. Furio Damiani: ______________________



Resumo

A tecnologia dos processadores tem crescido rapidamente nos últimos anos. Em contrapartida, o ensino de arquitetura de computadores tem dificuldade de acompanhar esta evolução. Os livros textos e as aulas ainda utilizam de recursos estáticos que precisam de longas explicações. Esta dinâmica torna-se incompatível, inclusive, com as experiências, como usuários, de muitos alunos.

Este trabalho propõe o desenvolvimento de um ambiente computacional (*framework*) para apoio ao ensino de arquitetura de processamento, denominado MODPRO. A ideia é dispor de módulos que possam ser interligados formando diversas possíveis estruturas de processamento. Desta maneira, o professor pode desenvolver junto aos alunos e expor de forma visual (utilizando de animações) desde componentes básicos até estruturas de processamento mais avançadas.

O MODPRO é composto por um simulador, denominado SIMPRO o qual exibe de forma animada, passo a passo, ou em tempo real, o fluxo de dados e de sinais dentro da estrutura de processamento estudada. O SIMPRO foi desenvolvido em linguagem Javascript e utiliza recursos de *Cascading Style Sheets*, podendo, ainda, ser acessado pela web. O MODPRO ainda é composto por um emulador, chamado EMUPRO que contém os mesmos módulos do SIMPRO. O seu diferencial está relacionado ao fato de ter sido totalmente desenvolvido em hardware, utilizando a ferramenta QUARTUS II da Altera. Com este recurso, os alunos podem, em laboratório, validar a estrutura desenvolvida em classe.

Por serem modulares, tanto o simulador SIMPRO quanto o emulador EMUPRO permitem que novos recursos (módulos) possam ser adicionados, permitindo assim, o ensino e o estudo de diferentes estruturas de processamento.

Palavras-chave: Ensino de Arquitetura de Computador; Simulação e Emulação de Processador; FPGA.

Abstract

The processor technology has grown rapidly in recent years. In constrast, the teaching of computer architectures has difficulties to follow such evolution. The books and the classes still use static resources which need long explanations. This dynamic becomes incompatible with the experiences desirable for most students.

This work proposes the development of a computational environment (framework) to support the teaching of processing architecture, which is called MODPRO. The idea is to have modules that can be connected together forming several possible processing structures. Therefore, the professor can develop different scenarios with the students and expose in a visual way (using animation features), from basic components to more advanced processing structures.

The MODPRO consists of a simulator, called SIMPRO which displays (in an animated form), step by step or in real-time, the data flow and the signal processing within the structure being studied. The SIMPRO was developed in JavaScript language, uses Cascading Style Sheets and can be accessed via web. The MODPRO also consistes in an emulator called EMUPRO, which contains the same modules of SIMPRO. Its differential is related to the fact that it has been developed entirely in hardware, using the development environment QUARTUS II from Altera. Basically, with the features of MODPRO, the students can validate, in laboratory, the frameworks and the processing structures developed during the classes.

Because they are modular, both the simulator SIMPRO and the emulator EMUPRO allow the addition of new features (modules), besides allowing the teaching and the study of different processing structures.

Keywords: Computer Architecture Teaching; Processor simulation and emulation; FPGA.

Agradecimentos

A Deus, pela minha vida, pela minha família e por me ensinar que a persistência é sem dúvida, o caminho para uma grande conquista.

Ao meu querido orientador, Prof. José Raimundo de Oliveira, pela confiança, dedicação, orientação, pelos ensinamentos e acima de tudo, pela humildade e por ser um grande exemplo de Professor. Serei eternamente grata.

Aos meus queridos pais, Leda e Attílio, razão do meu viver, pelo carinho, amor, apoio, por acreditarem em mim e não me deixarem nunca desistir. Tudo o que tenho, e o que sou devo à vocês. Obrigada por este amor incondicional, amo vocês.

Aos meus irmãos, Edivane e Edivar e minha cunhada Raquel, por vibrarem comigo, me apoiarem e estarem sempre ao lado, meu muito obrigada.

Aos meus sobrinhos, Laura e Leonardo, meus grandes amores, que encheram minha vida de alegria e amor.

Aos meus queridos amigos, que me acompanharam por todos esses anos, Wilfredo Puma, Rafael Pasquini, Tatiana Pazeto, Adler Silva, Jeremias Machado, Leandro Monteiro, Antonio Carlos Rodrigues, Letícia Ritnner, Tatiane Bonfim e em especial a Dra. Margareth Brigante (*in memoriam*) obrigada de coração, por tudo o que vocês fizeram por mim, pela amizade e pelos ótimos momentos compartilhados juntos.

Aos demais amigos e colegas de pós-graduação, com quem eu tive o privilégio de conviver. Em especial, meus amigos de laboratório André Delai e Filipe Fazanaro, por todo apoio e ajuda incondicional, que vocês dedicaram à mim.

Aos funcionários da Elétrica, Sr. Hélio, Noêmia, Edson, Juraci, em especial à Gislaine, pela dedicação, e a todos, que sempre me trataram com muito carinho e sempre estiveram à disposição para tudo o que eu precisei.

Agradeço à todos da comunidade FEEC, desde a diretoria e aos funcionários responsáveis pela limpeza, que sempre se preocuparam em proporcionar um ambiente agradável para os estudos.

A Nina, minha cachorrinha, minha companheira fiel de todos os momentos.

Aos meus queridos e amados pais, Leda e Attílio, meus irmãos Edivar e Edivane, minha cunhada Raquel e meus sobrinhos Laura e Leonardo

Sumário

Lista de Figuras xii

Lista de Tabelas xiv

Glossário xv

1 Introdução 1

1.1 Motivação 1

1.2 Objetivos 2

1.3 Organização do Trabalho 2

2 Conceitos Básicos 4

2.1 Arquitetura de Computadores 4

2.1.1 Arquitetura de *Princeton* 4

2.1.2 Arquitetura de *Harvard* 6

2.2 Computadores Paralelos 7

2.2.1 Processadores *Multicore* 9

2.3 Processamento Digital de Sinais 10

2.4 Unidade de Controle 11

2.5 Simulação e Emulação de Processadores 12

2.5.1 Simulador 13

2.5.2 Emulador 13

2.6 Resumo do capítulo 14

3 O Ambiente Computacional Proposto 15

3.1 O Ambiente ModPro 15

3.2 Unidade de controle 17

3.2.1 RI - Registrador de Instrução 17

3.2.2 Decodificador de Instruções 17
3.2.3 LUT - *Look-up Table* 17
3.2.4 Registrador de estado 18
3.2.5 Sistema de relógio 18
3.2.6 MIR - Registrador de micro-instrução 18
3.3 Operadores 18
3.3.1 Unidade Lógica e Aritmética - **ULA** 19
3.3.2 Multiplicador/Divisor 19
3.4 Registradores 19
3.4.1 Registradores contadores 19
3.4.2 Registrador de passagem 19
3.4.3 Registrador de propósito geral 20
3.5 Barramentos 20
3.5.1 Barramento Bidirecional 20
3.5.2 Barramento Unidirecional 20
3.6 Memória 20
3.7 Resumo do capítulo 21

4 Exemplos de Simulação e Emulação 22

4.1 Desenvolvimento do SIMPRO 22
4.1.1 Conjunto de Instruções 23
4.1.2 Memória Externa 24
4.1.3 Decodificador 24
4.1.4 Conjunto de Sinais de Controle 25
4.1.5 Execução de uma Instrução 25
4.2 Simulação da Arquitetura de *Princeton* no SIMPRO 27
4.2.1 Exemplo de Execução de uma Instrução 29
4.3 Simulação da Arquitetura de *Harvard* no SIMPRO 32
4.4 Simulação da Unidade MAC no SIMPRO 33
4.5 Desenvolvimento do EMUPRO 34
4.5.1 Componente Registrador 35
4.5.2 Componentes da Unidade Lógica e Aritmética 37
4.5.3 Multiplexador 39
4.5.4 Tratamento de Exceção 39
4.5.5 Componente Decodificador 40
4.5.6 Componente *Look-Up Table* 41

4.5.7 Componente Memória Externa 41
4.6 Emulação da Arquitetura de Princeton no EMUPRO 42
4.7 Resumo do capítulo 43

5 Conclusão 44
5.1 Trabalhos Futuros 45
5.1.1 Exemplo de Arquitetura Paralela no ModPro 45
5.1.2 Interface para o SIMPRO 47

Referências Bibliográficas 48

Lista de Figuras

2.1 Arquitetura de *Von Neumann*. 5
2.2 Execução de uma instrução na arquitetura de *Princeton*. 6
2.3 Ilustração da arquitetura de *Harvard*. 7
2.4 Execução de uma instrução na arquitetura de *Harvard*. 7
2.5 Arquitetura SISD. 8
2.6 Arquitetura SIMD. 8
2.7 Arquitetura MISD. 9
2.8 Arquitetura MIMD. 9
2.9 Unidade MAC. 11
2.10 Desenvolvimento com Emulador em circuito 14

3.1 Ilustração do *framework* proposto. 16

4.1 Simulação da Arquitetura de Princeton no SimPro. 28
4.2 Simulação da Instrução LOADI S - Conteúdo do PC. 29
4.3 Simulação da Instrução LOADI S - Leitura do Barramento pelo MAR. 30
4.4 Simulação da Instrução LOADI S - Leitura do Barramento pelo RI. 30
4.5 Simulação da Instrução LOADI S - Incremento do PC. 31
4.6 Simulação da Instrução LOADI S - Leitura do Barramento pelo MAR. 31
4.7 Simulação da Instrução LOADI S - Leitura do Barramento pelo Registrador S. . . . 32
4.8 Simulação da Arquitetura de Harvard no SimPro. 32
4.9 Simulação da Estrutura MAC no SIMPRO. 33
4.10 Registrador TFF. 35
4.11 Registrador UpDown. 36
4.12 Buffer de Leitura. 36
4.13 Somador e Subtrador. 37
4.14 Divisor. 38
4.15 Multiplicador. 38

4.16 Portas Lógicas AND e OR. 38
4.17 Multiplexador. 39
4.18 Circuito de Exceção do Acumulador. 40
4.19 Circuito de Carregamento do PC. 40
4.20 Decodificador de Instruções. 41
4.21 Look-Up Table. 41
4.22 Memória Externa. 42
4.23 Emulação da Arquitetura de *Princeton* no SIMPRO. 43

5.1 Simulação da Estrutura SIMD. 46
5.2 Simulação da Estrutura MIMD. 46

Lista de Tabelas

4.1 Definindo o conjunto de instruções. 24
4.2 Memória externa. 24
4.3 Decodificador de instruções. 25
4.4 Conjunto de sinais de controle. 26
4.5 Exemplo de uma instrução executada na LUT. 27

Glossário

PMAR - Registrador de Endereço de Memória de Programa (do inglês *Program Memory Address Register*)

BDM - *Background Debug Mode*

CPU - Unidade Central de Processamento (do inglês *Central Processing Unit*)

CSS - *Cascading Style Sheets*

DSP - Processamento Digital de Sinais (do inglês *Digital Signal Processing*)

FPGA - *Field Programmable Gate Array*

ICE - Emulador em Circuito (do inglês *In circuit emulator*)

JTAG - *Joint Test Action Group*

LUT - Tabela de Endereços (do inglês *Look-Up Table*)

MAC - Unidade de Multiplicação e Acumulação (do inglês *Multiply-Accumulate*)

MAR - Registrador de Endereço de Memória (do inglês *Memory Address Register*)

MIMD - Múltiplas Instruções, Múltiplos Dados (do inglês *Multiple Instruction Multiple Data*)

MIR - Registrador de Micro-Instruções (do inglês *Micro-Instruction Register*)

MIR - Registrador de micro-instruções (do inglês *Micro-Instruction Register*)

MISD - Múltiplas Instruções, Único Dado (do inglês *Multiple Instruction Single Data*)

PC - Contador de Programa (do inglês *Program Counter*)

RI - Registrador de Instrução (do inglês *Instruction Register*)

RI - Registrador de Instruções (do inglês *Joint Test Action Group*)

ROM - Memória somente de Leitura (do inglês *Read Only Memory*)

S - Apontador de Pilha (do inglês *Stack Pointer*)

SIMD - Única Instrução, Múltiplos Dados (do inglês *Single Instruction Multiple Data*)

SISD - Única Instrução, Único Dado (do inglês *Single Instruction Single Data*)

ULA - Unidade Lógica e Aritmética (do inglês *Arithmetic Logic Unit*)

Capítulo 1

Introdução

Com o objetivo de melhorar o ensino de estruturas de processamento e de prover aos alunos um conhecimento sobre as novas tecnologias, este trabalho propõe uma ferramenta didática para o ensino de um processador, utilizando um ambiente gráfico para ilustrar o funcionamento de uma determinada estrutura de processamento.

1.1 Motivação

A tecnologia investida nos computadores atualmente está se desenvolvendo muito rapidamente quando o assunto é o processador. Os processadores disponíveis no mercado possuem características únicas que atendem a diferentes públicos que não levam em consideração somente o seu custo/benefício mas, também, alguns outros fatores tais como a velocidade do processador, desempenho em se tratando do tempo para a execução de uma determinada tarefa e a quantidade de memória. Tais fatores podem ser considerados de extrema importância no momento de se adquirir um computador.

Com base nesse avanço tecnológico dos processadores, o ensino de arquitetura de computadores, exige uma constante atualização pois, conforme a tecnologia se desenvolve, novos conceitos acabam surgindo. Por outro lado, os professores tem dificuldade de acompanhar esta evolução pois os livros textos usados como referências e a forma como as aulas são ministradas utilizam de recursos estáticos que precisam de longas explicações. Esta dinâmica é incompatível, inclusive, com as experiências, como usuários, de muitos alunos.

O cenário que existe hoje dentro de uma sala de aula, mostra que os estudantes estão expostos a vários conceitos ou características que surgem, relacionadas à tecnologia de processamento, e começam a usufruir dessa tecnologia sem mesmo, antes, entender como esses conceitos são aplicados nas novas estruturas de processamento que estão surgindo.

Diante disso, a preocupação que existe é como apresentar os conceitos relacionados à estrutura de processamento para permitir uma rápida compreensão pelo aluno, bem como para permitir uma rápida adaptação a novas estruturas que emergem.

1.2 Objetivos

Dentro do contexto de ensino de arquitetura de computadores, o objetivo deste trabalho consiste em desenvolver um ambiente computacional, um *framework* didático, voltado para o ensino de arquitetura de processamento, sendo esse ambiente denominado de MODPRO. Este *framework* é formado por uma biblioteca composta de módulos específicos (tais como, registradores, ULA -Unidade Lógica e Aritmética (do inglês *Arithmetic Logic Unit*), memória ... etc) que podem ser conectados por um barramento para formar diversas estruturas de processamento. Assim sendo, o professor poderá mostrar, através de blocos gráficos e animação, os componentes básicos que compõe o processador e, também, estruturas de processamento mais avançadas.

O MODPRO é composto por um simulador, denominado SIMPRO, o qual foi desenvolvido em software utilizando a linguagem JavaScript e, através dele, é possível acompanhar de forma animada a execução de uma instrução realizada pelo processador, bem como os sinais de controle responsáveis por ativar a execução da instrução. O emulador, chamado EMUPRO, também faz parte do *framework*. Ele foi desenvolvido em hardware utilizando o ambiente QUARTUS II, versão 9.1 da Altera, sendo a emulação realizada através de um FPGA (*Field Programmable Gate Array*) [1].

1.3 Organização do Trabalho

De modo a facilitar a sua compreensão, este trabalho foi dividido como se segue.

No Capítulo 1, apresenta-se a introdução e a contextualização do trabalho desenvolvido, descrevendo o problema abordado e os respectivos objetivos.

No Capítulo 2, são abordados alguns conceitos básicos, aplicados a este estudo, tais como as arquiteturas *Princeton* e *Harvard*, Computadores paralelos, Processador Digital de Sinal, unidade MAC, arquitetura multicore, simuladores, emuladores e unidades de controle, destacando as principais características de cada um dos tópicos.

O Capítulo 3 faz uma descrição do *framework* proposto, denominado de SIMPRO, descrevendo o seu processo de desenvolvimento bem como o conjunto de componentes que fazem parte desse simulador.

No Capítulo 4, é descrito o desenvolvimento do emulador EMUPRO e seus principais componentes. Além disso, são apresentados alguns exemplos de simulação das arquiteturas de *Princeton*, de

Harvard e a unidade MAC e, também, exemplos de emulações aplicada à arquitetura de *Princeton*.

No Capítulo 5, são apresentadas as conclusões e as discussões relacionadas ao trabalho bem como sugestões de trabalhos futuros, sinalizando a simulação da arquitetura paralela e o desenvolvimento de uma interface para o SimPro.

Capítulo 2

Conceitos Básicos

Este capítulo introduz os principais conceitos utilizados nos demais capítulos deste trabalho. São aqui apresentadas as principais características das arquiteturas conhecidas como de *Princeton* e de *Harvard*, identificando suas principais diferenças. Uma introdução à classificação de Flynn, que ordena os sistemas de computadores em categorias, será feita. Esta classificação é muito utilizada para caracterizar os sistemas com capacidade de processamento paralelo. Também são introduzidas algumas características da arquitetura multicore (quando se utiliza dois ou mais processadores em um único *chip*), muito utilizada nos recentes processadores comerciais. Outro conceito aqui destacado é a Unidade de Multiplicação e Acumulação (MAC)(do inglês *Multiply-Accumulate*) presente em todos processadores digitais de sinais (DSP) do mercado. Um destaque especial, é dado à estrutura de controle da unidade central de processamento. Por último, serão discutidos os conceitos de simulação e emulação de sistemas de computadores.

2.1 Arquitetura de Computadores

A arquitetura de *Princeton* e a arquitetura *Harvard* consistem nas duas principais classes de arquiteturas de computadores existentes. A mais importante diferença está relacionada na maneira como a memória é acessada. Ao longo dessa seção, será realizada uma breve descrição sobre estas arquiteturas.

2.1.1 Arquitetura de *Princeton*

A arquitetura de *Princeton*, também conhecida como máquina de *Von Neumann*, ainda está presente na maioria dos computadores digitais. Esta arquitetura é composta por cinco unidades básicas: a memória principal, a unidade lógica e aritmética, a unidade de controle, e o equipamento de entrada

e saída [2]. Na Figura 2.1, tem-se uma ilustração destas unidades e suas interconexões.

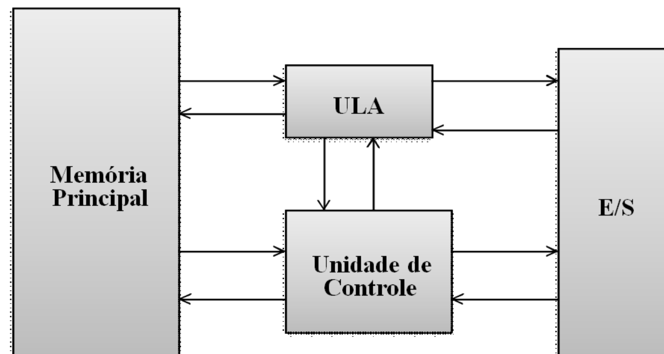


Fig. 2.1: Arquitetura de *Von Neumann*.

As unidades básicas da arquitetura *Princeton* podem ser descritas da seguinte maneira [3]:

- A memória principal é responsável por armazenar os dados e as instruções;
- A *ULA* realiza operações sobre os dados.
- A unidade de controle é a responsável pela interpretação e execução das instruções na memória;
- O equipamento de entrada e saída (E/S) é acionado pela unidade de controle.

Von Neumann definiu alguns pontos para esta arquitetura [3]:

- O computador deve conter uma unidade especializada para desempenhar as operações aritméticas de adição, subtração, divisão e multiplicação, que constitui a unidade lógica e aritmética;
- As operações podem ser executadas sequencialmente, através da unidade de controle;
- Qualquer dispositivo que precisar executar sequências de operações longas, precisa ter uma memória associada;
- O dispositivo precisa ter unidades para transferir informações de entrada e saída.

A característica básica da arquitetura de *Princeton* é que os dados e as instruções compartilham a mesma memória sendo acessada por um único barramento que também é utilizado para a transferência de dados e busca de instruções. Pelo fato de utilizarem o mesmo barramento, a transferência e a busca devem ser escalonadas e, consequentemente, não podem ser executadas ao mesmo tempo [4, 5, 6].

O ciclo de busca e execução é realizado através da CPU, que obtém a instrução da memória e a decodifica para identificar qual operação será realizada, executando a respectiva instrução. A instrução é composta por duas partes: OPCODE, que especifica a operação que será executada, e o

operando que informa quais dados serão operados [4].

A execução de uma instrução é realizada através (1) de sua busca na memória, (2) da decodificação realizada através da unidade de controle e (3) da execução da instrução. Quando ocorrer uma execução que requer dados (os quais possam ser lidos ou escritos na memória), esses somente serão obtidos após a conclusão da instrução em execução [4]. Baseando-se nisso, é possível considerar o conjunto de instruções de acesso à memória representadas por LOAD e STORE os quais são ilustrados na Figura 2.2.

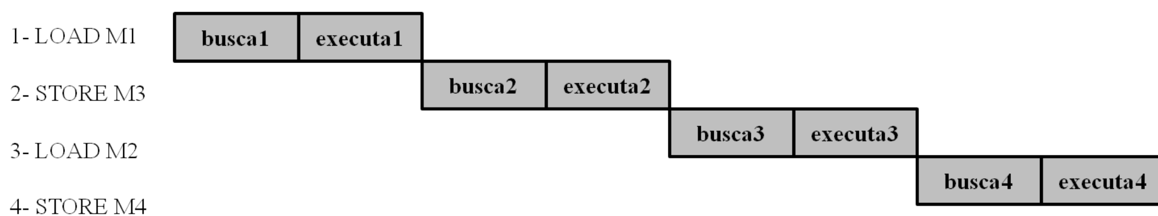


Fig. 2.2: Execução de uma instrução na arquitetura de *Princeton*.

A instrução STORE M3, somente poderá ser executada após a completa execução da instrução anterior, LOAD M1. Dessa forma, fica claro que as instruções são executadas sequencialmente na arquitetura de *Princeton*, ou seja, somente quando a instrução atual concluir sua execução é que a próxima poderá iniciar o ciclo de busca e execução.

A arquitetura de *Princeton* apresenta um gargalo entre a CPU - Unidade Central de Processamento (do inglês *Central Processing Unit*) e a memória, devido ao fato de os dados e as instruções compartilharem a mesma memória e, além disso, utilizar o mesmo barramento o que acarreta em uma diminuição no tráfego (*throughput*) dos dados [4].

2.1.2 Arquitetura de *Harvard*

A arquitetura de *Harvard* foi criada na Universidade de Harvard dentro do primeiro projeto de computador, denominado Harvard Mark I, o qual foi coordenado por Howard Aiken em 1944, em parceria da IBM e da marinha norte americana. Os protótipos iniciais faziam uso de válvulas e suas operações internas eram controladas por relés [4].

A principal característica da arquitetura de *Harvard* consiste na existência de duas memórias distintas, sendo uma aplicada às instruções e a outra aos dados [4, 6, 5]. Pelo fato de possuírem barramentos distintos, torna-se possível o acesso simultâneo, permitindo realizar a busca de instruções e de dados simultaneamente [4]. Na Figura 2.3, tem-se uma ilustração dessa arquitetura.

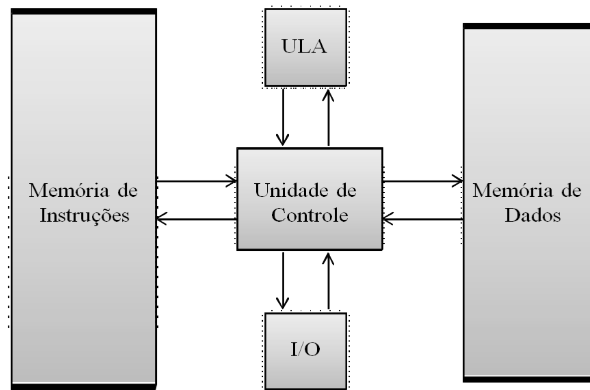


Fig. 2.3: Ilustração da arquitetura de *Harvard*.

O ciclo de busca de uma instrução e o de execução de outra instrução podem ser sobrepostos aplicando-se o conceito de *pipelining*, ou seja, o processador desenvolve um ciclo de execução enquanto realiza a busca da próxima instrução que será executada. A Figura 2.4 mostra uma sequência de execuções de instruções na arquitetura de *Harvard*.

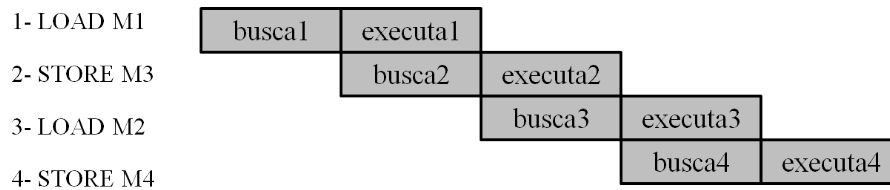


Fig. 2.4: Execução de uma instrução na arquitetura de *Harvard*.

Com base na Figura 2.4, considere o conjunto de instruções de acesso à memória LOAD e STORE. Observe que enquanto a instrução LOAD M1 está executando, a instrução seguinte, STORE M3 pode iniciar seu ciclo de busca e execução.

2.2 Computadores Paralelos

O processamento paralelo tem o objetivo de dividir as tarefas de modo que os processadores trabalhem de forma sincronizada, sendo que cada um executa uma tarefa. Baseando-se na classificação de computadores paralelos proposta por Flynn [7] (que observou o paralelismo nos fluxos de instruções e de dados), os computadores podem ser agrupados em quatro categorias [8] discutidas a seguir.

SISD - Única Instrução, Único Dado (*Single Instruction Single Data*)

Esta categoria é a mais simples e representa as máquinas clássicas de Von Neumann ou o monoprocessador. Existe um único fluxo de instruções e um único fluxo de dados, além de um único processador que executa uma única sequência de instruções que operam nos dados armazenados em uma única memória. Nesta categoria um único dado é operado por vez [7, 3], como pode ser observada na Figura 2.5.

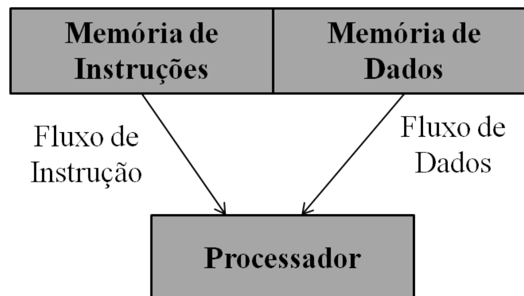


Fig. 2.5: Arquitetura SISD.

SIMD - Única Instrução, Múltiplos Dados (*Single Instruction Multiple Data*)

A categoria SIMD mostrada na Figura 2.6, possui uma única instrução que é executada em um conjunto diferente de dados e por diferentes processadores [3]. Cada processador possui sua própria memória de dados mas compartilham a mesma memória de instruções, ou seja, todos executam a mesma instrução, porém sobre diferentes dados, e possuem uma única unidade de controle, para busca e execução de instruções [8].

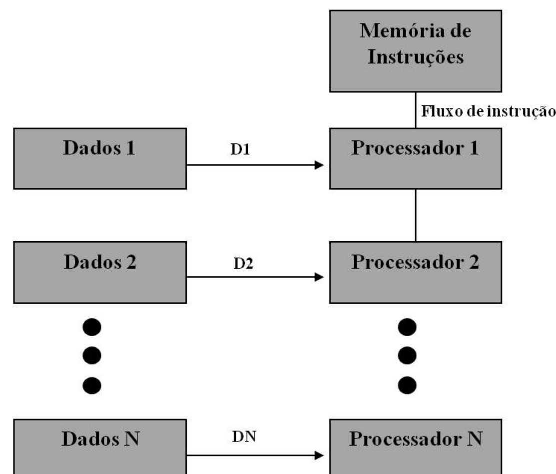


Fig. 2.6: Arquitetura SIMD.

MISD - Múltiplas Instruções, Único Dado (*Multiple Instruction Single Data*)

Uma sequência de dados é enviada para um conjunto de processadores, sendo que cada um executa uma sequência de diferentes instruções [3]. A Figura 2.7 mostra a arquitetura MISD.

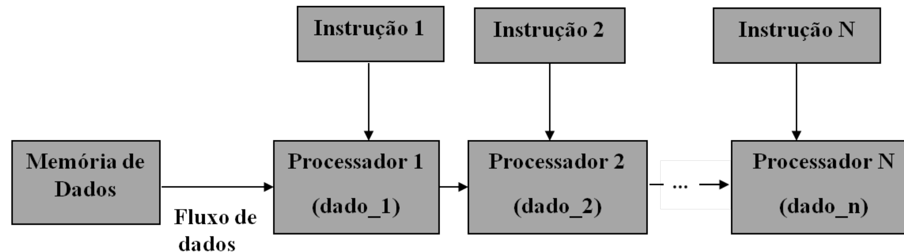


Fig. 2.7: Arquitetura MISD.

MIMD - Múltiplas Instruções, Múltiplos Dados (*Multiple Instruction Multiple Data*)

Um conjunto de processadores, representam a categoria MIMD sendo que cada um busca suas próprias instruções e executa sobre seus próprios dados [8]. Essa arquitetura pode ser observada na Figura 2.8.

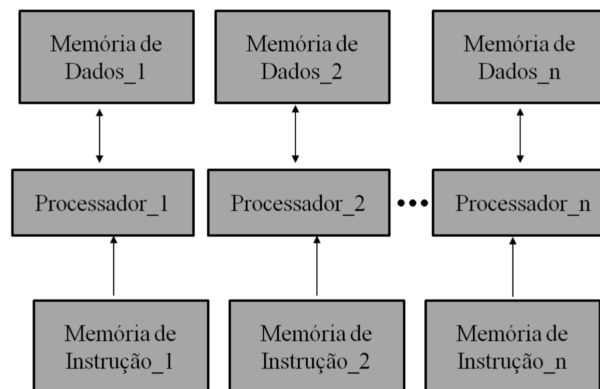


Fig. 2.8: Arquitetura MIMD.

2.2.1 Processadores *Multicore*

Os processadores *multicore* estão sendo aceitos na maioria dos segmentos da indústria, incluindo DSP. As implementações de processadores *multicore* são numerosas e diversas. Os projetos variam de máquinas com multiprocessadores convencionais, até projetos que consistem de um grande número de unidades lógicas e aritméticas (ULAs) programáveis [9].

A tecnologia *multicore* consiste em um sistema de computador que contém múltiplos processadores ou núcleos (*cores*) que fazem parte de um único *chip* [10, 3]. O sistema operacional trata cada um desses núcleos, como se fosse um processador diferente, com seus próprios recursos de execução como memória cache, registradores, ULA e unidade de controle, podendo processar várias instruções simultaneamente [10, 3]. O uso de vários núcleos possibilita executar instruções em paralelo e também diminui o problema do gargalo que causa atrasos e congestionamentos dos dados [10].

O objetivo de um sistema *multicore* é permitir maior utilização de paralelismo no nível de *threads*, que estão cada vez mais sendo utilizadas pelos projetos de software, para aplicações como multimídia e o amplo uso de ferramentas de visualização [10].

A principal vantagem de sistemas *multicore* é que o aumento de desempenho, pode estar relacionado ao número de processadores ao invés da frequência [9].

As arquiteturas *multicore* apresentam a possibilidade de distribuir as diferentes tarefas para os vários núcleos, obtendo maior eficácia (*troughput*) do sistema e um melhor desempenho por parte dos aplicativos mesmo que vários deles estejam executando simultaneamente [11].

2.3 Processamento Digital de Sinais

O processamento digital de sinais (DSP), (do inglês *Digital Signal Processing*), é uma área da ciência e engenharia que tem crescido muito nos últimos anos. Este crescimento é o resultado do avanço significativo na tecnologia do computador digital e fabricação de circuitos integrados. Os computadores digitais e o hardware digital associado nas últimas décadas eram grandes e seu custo era alto, como consequência, seu uso limitou-se para cálculos científicos e aplicações de negócios [12].

O DSP foi criado no início da década de 80, pelas principais empresas de eletrônicos, tais como a Texas Instruments, Analog Devices e Motorola, tornando-se em pouco mais de uma década um dos componentes mais importantes em eletrônica, sendo hoje a parte fundamental de muitos equipamentos de diversas áreas da indústria [13]. Ele surgiu com o propósito de se criar um microprocessador com uma arquitetura desenvolvida para realizar operações específicas necessárias ao processamento digital de sinal. Hoje, tem-se um produto que engloba, em um único *chip*, tecnologia suficiente para realizar praticamente qualquer tipo de processamento e análise de dados e sinais [13].

O DSP é um processador programável de propósito geral, que processa um sinal digital, e que possuem um custo razoável e um tempo eficiente para implementar algoritmos de processamento de sinal digital. A maioria dos DSPs são baseados na arquitetura de *Harvard*[14], que possui barramentos distintos para dados e para instruções e também possuem uma unidade de multiplicação paralela e muito rápida, denominada MAC, a qual é responsável pela execução de uma operação de

multiplicação em um único ciclo de clock [15, 14].

Por outro lado, há uma desvantagem que inclui a necessidade de hardware adicional como interface para portas I/O, conversores de dados analógico-digital e digital-analógico e unidades de memória interna. Além disso, devido à sua generalidade, os circuitos DSP são eficientes em relação à velocidade computacional quando comparado ao hardware dedicado para uma determinada aplicação [15].

Unidade de Multiplicação e Acumulação

O DSP é composto por uma unidade de multiplicação e acumulação, conhecida também por MAC. Esta unidade é projetada para resolver operações de cálculo para multiplicações de vetores e matrizes [8], exigida em algoritmos de filtragem digital e processamento de sinais variantes no tempo [15]. O objetivo do MAC é multiplicar dois operandos, a e b, e adicionar o resultado do produto no acumulador, como pode ser observado na Figura 2.9. No próximo *loop*, ou seja, na entrada dos novos operandos, o valor do acumulador, resultante das multiplicações anteriores dos operandos, é somado e acumulado ao resultado da multiplicação dos novos operandos.

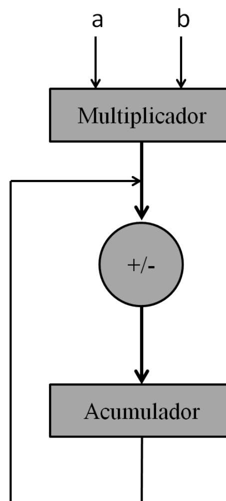


Fig. 2.9: Unidade MAC.

2.4 Unidade de Controle

A unidade de controle tem a função de ativar os sinais de controle responsáveis pela execução de uma instrução. Ela tem como entrada o código da instrução a ser executada. Este código fica armazenado durante o ciclo de execução da instrução num registrador, chamado RI - Registrador

de Instruções (do inglês *Instruction Register*). A partir desse código, a unidade de controle gera sequencialmente os sinais de controle que realizam as micro-operações necessárias à realização da instrução. Por micro-operação entende-se, por exemplo, o gatilho de um registrador, a ativação de um barramento ou o apagamento (*reset*) de um registrador. A unidade de controle pode ser de dois tipos: *hardwired* ou microprogramada.

A unidade de controle do tipo *hardwired*, também conhecida como implementada por hardware, é um circuito lógico baseado em uma máquina de estado que implementa os estados necessários para ativar todos os sinais de controle de um conjunto de instruções [3, 16]. Este tipo de unidade de controle tem a vantagem de ocupar menos espaço que a outra alternativa. No entanto, ela é pouco flexível para, por exemplo, incluir nova instrução. Neste caso o projeto de todo o circuito lógico deverá ser refeito.

A unidade de controle microprogramada é especificada por um microprograma que consiste de uma sequência de microinstruções em uma linguagem de microprogramação [3]. Este microprograma pode ser armazenado numa ROM - Memória somente de Leitura (do inglês *Read Only Memory*), chamada de memória de microprograma ou micromemória, que em cada posição armazena um vetor com todos os sinais de controle para os componentes do processador [16]. Cada sinal de controle ocupa um *bit* da palavra armazenada. Em cada passo de execução de uma instrução, uma nova posição desta memória é acessada e seus *bits* definem o estado de cada sinal de controle. Durante cada passo de execução de uma instrução, esta palavra é mantida estável num registrador, chamado de MIR - Registrador de Micro-Instruções (do inglês *Micro Instruction Register*). A memória de microprograma é também referenciada como uma LUT - Tabela de endereços (do inglês *Look-Up Table*). A unidade de controle microprogramada possui a vantagem de ser flexível para incluir novas instruções, ou mesmo, otimizar as existentes. No entanto, ela pode necessitar de uma área física maior para sua implementação.

Para um estudo aprofundado sobre a estrutura de processamento e sobre os tipos de unidades de controle, vide [3] ou [16].

2.5 Simulação e Emulação de Processadores

Segundo Michaelis [17] o termo **Simular** significa, dentre outros, ***Arremedar ou imitar***. Ainda segundo este dicionário, o termo **Emular** significa, dentre outros, ***comportar-se como alguma outra coisa***. Ou seja, de forma livre pode-se dizer que os dois termos têm um significado muito próximo, de "imitar". Em computação, uma distinção é feita dependendo de como esta "imitação"é implementada. Originalmente, diz-se simular, quando esta "imitação"é realizada em software. Diz-se emular, quando ela é implementada por hardware.

2.5.1 Simulador

A simulação é utilizada para descrever e analisar o comportamento de um sistema e, usualmente, é realizada através de softwares de desenvolvimento (tais como o QUARTUS II da Altera [18]) e tem a função de verificar computacionalmente a operação de um processo ou de um sistema implementado fisicamente [19]. O resultado de uma simulação ajuda a definir o *layout* físico de um sistema, limitando a funcionalidade e o controle do sistema [20].

Existem alguns simuladores como o SIMULINK, voltado para simulação de sistemas dinâmicos [21], o SPICE, que permite analisar o comportamento elétrico de um circuito [22] e o ARENA que é um ambiente gráfico de simulação que ajuda a demonstrar, prever e medir estratégias do sistema para obter um desempenho eficaz, eficiente e otimizado [23]. Estes simuladores foram desenvolvidos com o intuito de poder realizar qualquer tipo de simulação, para qualquer área de atuação. Contudo, dependendo do sistema, a simulação pode se tornar complexa e levar muito tempo para ser concluída.

2.5.2 Emulador

Durante muito tempo o desenvolvimento de sistemas de programas para microcomputador utilizou o que era chamado de *ICE* - Emulador em circuito (do inglês *In circuit emulator*). Eram equipamentos fornecidos por fabricantes de microprocessadores e por alguns fabricantes de instrumentos que permitiam o teste e a depuração do programa juntamente com o teste e a depuração do circuito em desenvolvimento. Estes equipamentos dispunham de diversos recursos como grande quantidade de memória, recursos de interface com o usuário, software de desenvolvimento de programas (compiladores e montadores) e recursos de depuração. Em geral, os recursos de depuração destes equipamentos permitiam ao programador, acompanhar a execução de seu programa de forma controlada. Ou seja, estabelecendo pontos de parada (*breakpoints*) em determinadas posições do programa e nestes pontos observar, ou se necessário modificar os conteúdos de determinados elementos armazenadores (posições de memória ou variáveis do programa e registradores) [24, 25].

Os ICE's se ligavam ao circuito em desenvolvimento através do que era chamado de "cordão umbilical"que tinha em uma de suas extremidades um conector com pinos compatíveis com o microprocessador utilizado no circuito. Esta extremidade era ligada ao soquete do microprocessador no circuito em teste e rodava o programa residente na memória do emulador, observe a Figura 2.10. Após toda a depuração, o cordão umbilical era retirado e o microprocessador colocado em seu soquete juntamente com a memória de programa (em geral, EPROM) com o programa depurado.

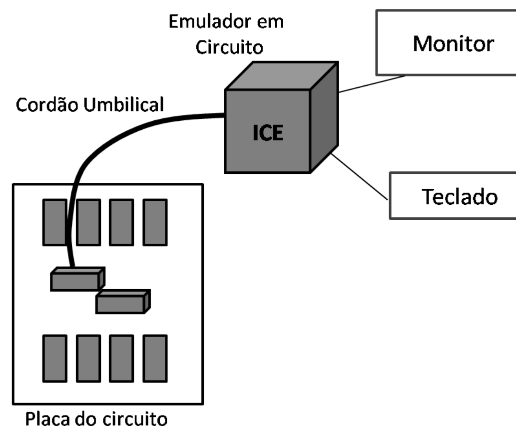


Fig. 2.10: Desenvolvimento com Emulador em circuito

Hoje em dia, os emuladores em circuito não são mais usados. Dois são os principais motivos. Primeiro, o surgimento de simuladores cada vez mais sofisticados, que permitem um desenvolvimento em condições mais próximas do processador alvo. Também, graças ao desenvolvimento da microeletrônica, hoje em dia os processadores têm integrado em seus chips diversos recursos que permitem a depuração no próprio circuito final do desenvolvimento. São exemplos destes recursos o padrão IEEE 1149 JTAG - *Joint Test Action Group* [26] e o BDM - *Background Debug Mode*, da Freescale [27].

2.6 Resumo do capítulo

Neste capítulo, foram apresentados, de forma resumida, diversos conceitos importantes no estudo de arquitetura de computadores. Muitos destes conceitos poderiam necessitar de um capítulo inteiro para descrevê-los de forma completa. Aqui não se pretendeu esgotar cada um destes conceitos e sim apresentá-los de forma suficiente para o entendimento do seu uso no restante deste trabalho. Para aqueles interessados em um estudo mais aprofundado sobre os tópicos aqui abordados recomenda-se as referências [2, 3]. Dentre os conceitos aqui apresentados, destaca-se o de microprogramação. Este conceito é fundamental para este trabalho, pois ele foi utilizado para a implementação da unidade de controle descrita no capítulo 3.

Os conceitos de simulação e emulação aqui apresentados são muito amplos. Nos próximos capítulos deste trabalho, os termos "simulador"e "emulador"são utilizados de forma a distinguir o que foi implementado em software do que foi implementado em hardware.

Capítulo 3

O Ambiente Computacional Proposto

Um processador é composto por um conjunto de elementos básicos que, trabalhando juntos, realizam a execução de instruções. Esses elementos são os registradores, unidades de operação lógica e aritmética, memória, unidade de controle e barramentos. Dentro deste contexto, este capítulo propoe um ambiente computacional *framework*, voltado para o ensino de estruturas de processamento.

3.1 O Ambiente ModPro

O ambiente computacional proposto é denominado MODPRO. Trata-se de um recurso didático para ser usado no ensino de arquitetura de processamento de dados, ou seja, tem como objetivo o ensino do funcionamento de processadores, bem como, de cada um de seus componentes. A idéia é que através de um conjunto de módulos interligáveis seja possível formar diversas estruturas de processamento. Desta forma, o professor pode desenvolver, em sala de aula, junto com o aluno e mostrar de forma visual com animação, desde componentes básicos do processador até as estruturas de processamento que estão emergindo no mercado. Este ambiente é composto por um simulador, denominado SIMPRO, e por um emulador, chamado EMUPRO, como pode ser observado na Figura 3.1. Observe que os termos simulador e emulador são usados aqui de forma livre, somente para distinguir que o SIMPRO é implementado em software e o EMUPRO é realizado em hardware.

O MODPRO é composto por um conjunto de elementos básicos que são descritos tanto no SimPro quanto no EmuPro. No SIMPRO, estes elementos são representados, principalmente, de forma simbólica, no EMUPRO, eles são descritos através de módulos funcionais no ambiente de desenvolvimento QUARTUS II da Altera.

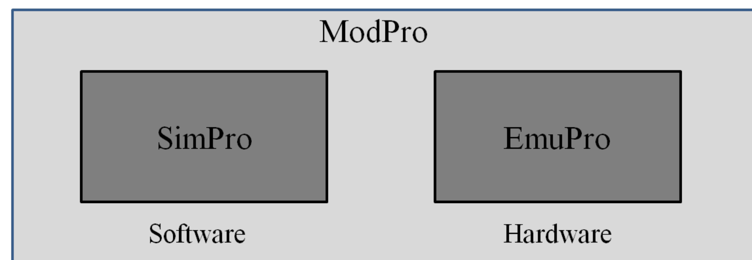


Fig. 3.1: Ilustração do *framework* proposto.

O SIMPRO é um conjunto de recursos gráficos (módulos) que permite ao professor ensinar de forma animada, passo a passo os sinais de controle e todo o fluxo de dados durante a execução do ciclo de uma instrução pelo processador. Os módulos do SIMPRO permitem ao professor moldar/criar uma determinada arquitetura de componentes, possibilitando a inclusão de novas estruturas de processamento. O SimPro tem o compromisso de ficar disponível na internet, onde poderá ser exercitado pelos alunos.

O EMUPRO, contém os mesmos módulos do SIMPRO, mas é totalmente desenvolvido em hardware, utilizando a ferramenta QUARTUS II da Altera [18]. Através deste recurso, os alunos podem, em laboratório, validar a estrutura desenvolvida em classe.

Para a elaboração do MODPRO foi considerado que cada um dos seus elementos básicos, ou módulo, possui sinais de controle que vêm de um módulo central de controle. Este módulo central de controle, aqui chamado de unidade de controle, é do tipo microprogramado. Assim, esta unidade de controle é comum a qualquer implementação de processador. O que varia da implementação de um processador para outro é a quantidade de sinais de controle que esta unidade gera e o microprograma nela residente.

Além da unidade de controle, os demais módulos disponíveis no MODPRO são classificados nos seguintes tipos:

- Operadores;
- Registradores;
- Barramentos;
- Memória.

As seções seguintes descrevem cada um dos elementos básicos disponíveis no MODPRO.

3.2 Unidade de controle

Este é o módulo mais importante de todo o MODPRO. Ele é comum a todas implementações de processadores. A unidade de controle do MODPRO é do tipo microprogramada, de forma que é possível microprogramar toda e qualquer sequência de micro-operações para realizar qualquer instrução.

A Unidade de Controle é composta por:

- RI - Registrador de Instruções;
- Decodificador de Instruções;
- LUT - *Look-up Table*;
- Registrador de estado;
- Sistema de relógio;
- MIR - Registrador de micro-instrução (do inglês *Micro-Instruction Register*)

3.2.1 RI - Registrador de Instrução

O registrador de instruções é a entrada da unidade de controle. Ele recebe, após um ciclo de busca de instrução, do barramento de dados interno, o código que representa a instrução a ser executada. Ele possui um único sinal de controle referenciado por um gatilho, que permite a entrada de dados para este registrador.

3.2.2 Decodificador de Instruções

O circuito de decodificação de instruções é responsável por converter o código da instrução a ser executada no endereço inicial na LUT, a partir de onde inicia a sequência de microoperações para a execução da instrução. No SIMPRO este circuito não é explicitamente representado, está dentro do bloco que representa a unidade de controle, dessa forma, ele é implementado por uma tabela de conversão. No EMUPRO também é implementado por uma tabela que é transformada em circuito pelo sistema Quartus II. Este módulo não possui sinais de controle.

3.2.3 LUT - *Look-up Table*

Este é o coração de todo o ambiente, é a memória de microprograma, responsável por armazenar todas as sequências de micro-operações de todas as instruções implementadas no processador desen-

volvido. Ela é uma memória, sendo que para cada um de seus endereços encontram-se associados ou são associados valores binários, que representam sinais para o controle dos componentes do projeto.

3.2.4 Registrador de estado

O registrador de estado sinaliza as condições finais da execução completa de uma instrução. Por exemplo, se após a execução de uma instrução um registrador foi levado a conter o valor zero, isto estará sinalizado num bit no registrador de estado. Estas condições podem ser utilizadas na implementação de instruções de desvio condicionado.

3.2.5 Sistema de relógio

O sistema de relógio da unidade de controle permite o funcionamento de todo ambiente em duas formas básicas:

Passo a passo: quando a cada ciclo de relógio o sistema para, o que permite uma visualização do estado de cada um dos módulos envolvidos na execução da presente instrução.

Tempo de execução real: quando o relógio do sistema roda livremente.

No SIMPRO, o sistema de relógio é implícito ao bloco da unidade de controle. Dele somente são visíveis dois botões, um que permite a seleção da forma de funcionamento (passo a passo) e outro que dispara o ciclo de relógio. No EMUPRO, o sistema de relógio utiliza o relógio da placa da FPGA, uma chave seletora, que permite a execução passo a passo, e um botão que dispara o ciclo de relógio.

3.2.6 MIR - Registrador de micro-instrução

O registrador de micro-instrução recebe em cada ciclo de relógio a palavra com os *bits* de controle de todos os módulos do processador. Este registrador mantém estes sinais de controle durante todo o ciclo de relógio.

3.3 Operadores

São chamados módulos operadores aqueles responsáveis pelas operações lógicas e aritméticas realizadas no processador. Estão disponíveis os seguintes módulos operadores:

- Unidade Lógica e Aritmética (**ULA**)
- Multiplicador/divisor

3.3.1 Unidade Lógica e Aritmética - ULA

A **ULA** é responsável por realizar as operações lógicas e aritméticas dos dados no computador. As operações lógicas estão relacionadas às operações *bit* a *bit* de **AND**, **OR**, **NOT** e **XOR**. Esta unidade também realiza as operações aritméticas básicas de adição e subtração.

3.3.2 Multiplicador/Divisor

Este módulo realiza as operações de multiplicação e divisão inteiras.

3.4 Registradores

Os registradores são memórias que armazenam temporariamente as informações que devem ser usadas na execução de uma instrução. Eles são formados por um conjunto de *flip-flops*, sendo que cada um é capaz de armazenar um *bit*. Existem vários tipos de registradores que fazem parte do MODPRO de forma que cada um é responsável por uma determinada função dentro do processador.

Estão disponíveis no MODPRO os seguintes tipos de registradores:

- Registradores contadores
- Registrador de passagem
- Registrador de propósito geral

3.4.1 Registradores contadores

Na biblioteca do MODPRO existe o chamado registrador contador. É um tipo de registrador com capacidade de contagem crescente ou decrescente. Em geral, ele é instanciado para realizar funções específicas dentro de uma estrutura de processamento. Exemplos de uso, Contador de Programa (PC), do inglês *Program Counter* e o Apontador de Pilha S, do inglês *Stack Pointer*.

3.4.2 Registrador de passagem

Na biblioteca do MODPRO está disponível ainda o chamado registrador de passagem. Este tipo de registrador é instanciado para funções de armazenamento temporário dentro de uma estrutura de processamento. Em geral, as instâncias deste tipo de registrador não são visíveis pelo jogo de instruções implementado. O nome deste tipo de registrador está associado ao fato de que o seu uso é, em geral, para interligar (ponto a ponto) somente dois módulos da estrutura de processamento

implementada. Exemplos de uso, MAR - Registrador de Endereço de Memória (do inglês *Memory Address Register*) e outros registradores temporários.

3.4.3 Registrador de propósito geral

O último tipo de registradores disponíveis na biblioteca do MODPRO é chamado de registrador de propósito geral. Este tipo é instanciado para formar os registradores que serão visíveis ao jogo de intruções, ou seja, que estão presentes nas instruções.

3.5 Barramentos

Entende-se por barramento um conjunto de fios paralelos os quais permitem a ligação entre os componentes. Eles são usados para transmitir dados, endereços e sinais de controle [2]. Estão disponíveis no MODPRO dois tipos de barramento: o bidirecional e o unidirecional.

3.5.1 Barramento Bidirecional

São módulos de interligação nos dois sentidos (leitura ou escrita). Exemplo do uso: na implementação do barramento interno de dados e no barramento de dados da memória externa.

3.5.2 Barramento Unidirecional

São módulos de interligação num único sentido. Exemplo do uso: na implementação do barramento de endereço e de controle da memória.

3.6 Memória

O MODPRO define um módulo de memória. O módulo de memória é útil para armazenar informações (dados e ou programas). Ela é composta por um conjunto de células. Cada célula tem associada um endereço, que pode ser referenciado por um programa [2].

Instâncias deste módulo podem ser usadas para a implementação de unidade de memória de dados, de programa e memórias cache.

3.7 Resumo do capítulo

Neste capítulo, foi descrita a proposta de um ambiente computacional para apoio ao ensino de estruturas de processamento. Este ambiente é formado por duas partes, pelo SIMPRO e pelo EMUPRO. Este ambiente define uma biblioteca de módulos que podem compor uma estrutura de processamento. Cada módulo desta biblioteca é descrito na forma simbólica, para uso no SIMPRO e na sua forma lógica operacional para uso no EMUPRO.

O SIMPRO foi desenvolvido usando a linguagem *JavaScript*, empregando recursos de CSS - *Cascading Style Sheets* e, permite ser acessado pela web. Seu uso está focado para fins didáticos, ou seja, para o ensino no curso introdutório de arquitetura de computadores.

O SIMPRO pode utilizar os componentes do MODPRO para compor uma determinada estrutura de processamento. Dessa forma, através de um conjunto de recursos gráficos, o SIMPRO permite ao professor mostrar de forma animada, passo a passo (ou em tempo de execução), os sinais de controle e todo o fluxo de dados durante o ciclo de uma instrução executada pelo processador. Com isso, os alunos podem acompanhar a execução de uma determinada instrução dentro da estrutura de processamento estudada.

O EMUPRO foi desenvolvido em hardware, usando a ferramenta QUARTUS II (versão 9.1) da Altera [18]. A sua função consiste em emular a sequência de um ciclo de instrução, usando um FPGA. O EMUPRO é utilizado para validar em laboratório, a estrutura de processamento apresentada aos alunos através do SIMPRO.

No próximo capítulo serão apresentados exemplos da aplicação do ambiente computacional MODPRO.

Capítulo 4

Exemplos de Simulação e Emulação

Este capítulo apresenta o desenvolvimento do SIMPRO e do EMUPRO bem como alguns exemplos de aplicação. Para o SIMPRO foram aplicados exemplos utilizando as arquiteturas de *Princeton*, de *Harvard* e a estrutura do MAC. O objetivo é mostrar como estas arquiteturas serão simuladas no SIMPRO. Para o exemplo no EMUPRO, foi utilizada a arquitetura de *Princeton*. A escolha das arquiteturas de *Princeton* e de *Harvard*, como exemplo de simulacão e também emulação, se deve ao fato delas serem mais conhecidas na literatura. Já a utilização da estrutura do MAC é justificada por ser considerada mais atual dentre as três.

4.1 Desenvolvimento do SIMPRO

O SIMPRO foi desenvolvido baseado em um processador de 8 *bits*, sendo composto por uma ULA e pelos seguintes registradores: Acumulador (A), registradores B, C, D, o registrador Temporário (T), o *Stack Pointer* (S), o Registrador de Instruções (RI), registrador X, o PC, e o MAR. Também faz parte do SIMPRO a memória externa composta por um conjunto de instruções e, tambem, o barramento de comunicação entre os registradores e a ULA.

Esse conjunto de componentes faz parte da biblioteca do *framework* e foi selecionada para compor o SIMPRO. As representações associadas a cada registrador foram definidas para serem usadas no SIMPRO, ou seja, para cada tipo de estrutura que for estudada, o professor poderá escolher os componentes desejados, nomeando-os de acordo com a estrutura ensinada.

A ULA tem o papel de realizar as operações aritméticas de adição, subtração, divisão e multiplicação e as operações lógicas AND e OR representadas respectivamente por & e |. Os resultados das operações realizadas pela ULA são armazenados no Acumulador (A).

Os registradores que fazem parte do SIMPRO são responsáveis por armazenar os valores dos operandos para que posteriormente estes sejam utilizados nas operações realizadas pela ULA.

O SIMPRO é composto por quatro arquivos importantes que identificam os dados que serão processados sendo, então, os responsáveis por todo funcionamento do processador. O projetista é o responsável por definir quais e quantas operações serão utilizadas no SIMPRO. Estes arquivos são descritos da seguinte maneira:

- o conjunto de instruções que o processador é capaz de executar;
- a memória externa;
- o decodificador;
- e a LUT.

4.1.1 Conjunto de Instruções

O conjunto de instruções contém todas as instruções que o processador poderá executar. Este arquivo é composto por números que foram definidos pelo projetista e que variam entre 0 a 255. Cada número representa o código da operação, denominado OPCODE, que identifica a operação que será executada. A cada OPCODE está associada uma instrução e, consequentemente, este número permite identificar qual instrução deverá ser executada.

Cada instrução tem associada uma abreviação, denominada mnemônico, que é usada pela linguagem *Assembly* para identificar a instrução que deverá ser executada [3]. Por exemplo, o mnemônico ADD é uma instrução que realiza a operação de adição.

Para exemplificar o conjunto de instruções, observe a Tabela 4.1. O OPCODE '00' representa a função que é identificada pelo mnemônico HALT cujo objetivo é parar o processamento. O OPCODE '01' está associado à função CLR A que tem o papel de zerar o acumulador. Já o OPCODE '02' representa a função INC A que irá incrementar o conteúdo do acumulador e '31' está relacionado à função LOADI S que realiza o carregamento imediato, ou seja, o conteúdo da próxima posição de memória para o registrador *stack pointer* (S). Baseado nisso, se define a tabela com o conjunto de instruções que o processador poderá desempenhar.

OPCODE	Instrução
00	HALT
01	CLR A
02	INC A
⋮	⋮
31	LOADI S

Tab. 4.1: Definindo o conjunto de instruções.

4.1.2 Memória Externa

A memória externa é composta pelo conjunto de instruções e pelos dados. Este arquivo é constituído por um *array* de 256 palavras que variam de $[0 \dots 255]$. Cada palavra representa uma posição (célula) na memória externa e o respectivo conteúdo de memória de cada posição indica o OPCODE associado a cada instrução. Para exemplificar, observe a Tabela 4.2.

Posição de Memória	Conteúdo da Memória/OPCODE
0	31
1	315
2	29
3	35

Tab. 4.2: Memória externa.

A posição '0' da memória externa tem como conteúdo o OPCODE '31'. O valor '31', na Tabela 4.1, que está relacionada ao conjunto de instruções, representa a instrução LOADI S a qual carrega o registrador S com o valor imediato, ou seja, o conteúdo da próxima posição da memória, que seria a posição '1', cujo conteúdo é '315'. A posição '2', possui o OPCODE '29' que representa a instrução LOADI D que carrega o registrador D com o valor imediato 35, que ocupa a posição '3' da memória.

As instruções que irão fazer parte da memória externa, bem como o OPCODE associado a cada uma das instruções, serão especificados pelo programador.

4.1.3 Decodificador

O decodificador tem a função de interpretar o OPCODE, indicando o endereço na LUT em que a instrução deverá iniciar sua execução. A Tabela 4.3 ilustra como é realizado este processo.

Decodificador	Endereço na LUT	Instrução
0	459	HALT
29	89	LOADI D
31	99	LOADI S

Tab. 4.3: Decodificador de instruções.

O OPCODE representa o conteúdo da memória externa, como mostrado na Tabela 4.2, e ocupa o campo Decodificador na Tabela 4.3. Considere, por exemplo, o OPCODE '31' da Tabela 4.2 correspondente ao endereço 99 na LUT. Isso significa que a instrução LOADI S terá início à sua execução no endereço 99, como pode ser observado na Tabela 4.3. Da mesma forma isso acontece com o OPCODE '29', que corresponde o endereço 89 na LUT. Esse procedimento é realizado para todo o conjunto de instruções que foram definidas pelo projetista.

4.1.4 Conjunto de Sinais de Controle

A Tabela 4.4 mostra todos os sinais de controle e a função de cada um deles no SIMPRO.

O arquivo correspondente à LUT contém informações de todos os sinais de controle que devem ser ativados para que uma determinada instrução possa ser executada.

Os sinais de controle BL, BZ, BG, LT, AZ e MQ fazem parte do registrador de estados. Estes sinais representam um conjunto de flags que indicam o estado dos resultados que saem do acumulador. O sinal DZ é um controle que faz parte do registrador D, indicando quando o resultado deste registrador é igual a zero. Quando qualquer um destes sinais de estados estiver ativado, o valor do PC será alterado.

4.1.5 Execução de uma Instrução

Baseado no conjunto de sinais de controle apresentados na Tabela 4.4, é possível acompanhar a execução de uma determinada instrução na LUT, como pode ser observado na Tabela 4.5. Para o exemplo, será executada a instrução LOADI S, que tem o OPCODE '31' e ocupa o endereço 99 na LUT.

Sinal de Controle	Definição
OP	Ativa/desativa a linha de micro-instrução válida
RA	Lê do acumulador
RB	Lê do registrador B
RC	Lê do registrador C
RD	Lê do registrador D
RS	Lê do registrador Stack Pointer S
RT	Lê do registrador temporário T
RX	Lê do registrador X
RP	Lê do registrador *Program Counter* P
RM	Lê do registrador de memória M
GA	Gatilha/escreve no acumulador
GB	Gatilha/escreve no registrador B
GC	Gatilha/escreve no registrador C
GD	Gatilha/escreve no registrador D
GT	Gatilha/escreve no registrador temporário T
GS	Gatilha/escreve no registrador Stack Pointer S
GX	Gatilha/escreve no registrador X
GP	Gatilha/escreve no registrador Program Counter P
BL	Verifica se o resultado é menor que zero
BZ	Verifica se o resultado é igual a zero
BG	Verifica se o resultado é maior que zero
LT	Verifica se o resultado do acumulador é menor que zero
AZ	Verifica se o resultado do acumulador é igual a zero
MQ	Verifica se o resultado do acumulador é maior que zero
GI	Gatilha/escreve no registrador de instruções I
GM	Gatilha/escreve no registrador de endereço de memória M
ME	$\bar{M}$*emory Enable*, a posição do endereço de memória do MAR é localizado na memória externa
IP	Incrementa o registrador *Program Counter* P
IA	Incrementa o acumulador
ID	Incrementa o registrador D
IS	Incrementa o registrador *Stack Pointer* S
IX	Incrementa o registrador X
DA	Decrementa o acumulador A
DD	Decrementa o registrador D
DS	Decrementa o registrador *Stack Pointer* S
DX	Decrementa o registrador X
DZ	Compara se o registrador D é igual a zero
CL	Zera o acumulador A
CC	Zera o contador da LUT

Tab. 4.4: Conjunto de sinais de controle.

Sinais de Controle								Endereço LUT	Decodificador	Instrução LOADI S
OP	GS	RP	RM	GM	ME	IP	CC			
1	0	1	0	0	0	0	0	99	31	(PC)->(bus);
1	0	1	0	1	0	0	0	100		(bus)->(MAR);
1	0	0	1	0	1	0	0	101		(mem[MAR])->(bus);
1	1	0	1	0	1	1	0	102		(bus)->(regS); (PC)+1 ->(PC);
1	0	0	0	0	1	0	1	103		e desvia para o ciclo de busca

Tab. 4.5: Exemplo de uma instrução executada na LUT.

Para este exemplo, somente os sinais de controle, necessários para a execução da instrução LOADI S é que serão mostrados na Tabela 4.5. Quando o sinal de controle for 1, então ele está ativado, se for 0 está desativado.

A instrução leva cinco linhas de micro-operações para ser executada, ela tem início no endereço 99 da LUT e finaliza sua execução no endereço 103. A primeira linha de micro-instrução indica que o valor do PC é colocado no barramento e, portanto, os sinais de controle ativos são a operação (OP), que indica se a micro-instrução é válida ou não para a instrução que está sendo executada. Se OP for igual a '1' a micro-instrução é válida e portando, será executada, caso contrário, se OP for igual a '0', a micro-instrução não é válida e portanto, não será executada. A leitura do PC é representada por RP. Na segunda linha, o (MAR) realiza a operação de leitura do barramento e armazena os dados temporariamente. Para isso, o sinal de controle GM é ativado. Observe que o sinal RP permanece ativo até que a escrita seja realizada pois, caso contrário, o valor seria perdido. Na terceira linha, o endereço apontado pelo MAR é localizado na memória externa e o conteúdo, deste endereço, é colocado no barramento de dados. Para isso, os sinais de controle ativos são RM e ME. A quarta linha indica que o valor que está no barramento de dados é armazenado pelo registrador S, através do sinal GS, e o PC é incrementado, através do sinal IP. Observe que o sinal ME continua ativo até que o registrador S realize a leitura do barramento. A última linha desvia para o ciclo de busca da próxima instrução, zerando o contador da LUT através do sinal de controle CC. Após essa execução, o endereço contendo a próxima instrução seria buscado para iniciar a execução da nova instrução.

4.2 Simulação da Arquitetura de *Princeton* no SIMPRO

Baseado no conjunto de recursos gráficos do SIMPRO, a arquitetura de *Princeton* é representada por um conjunto de blocos que representam os diversos tipos de componentes que compõem o pro-

cessador. Estes blocos, ao serem utilizados na simulação, foram nomeados como pode ser observado na Figura 4.1.

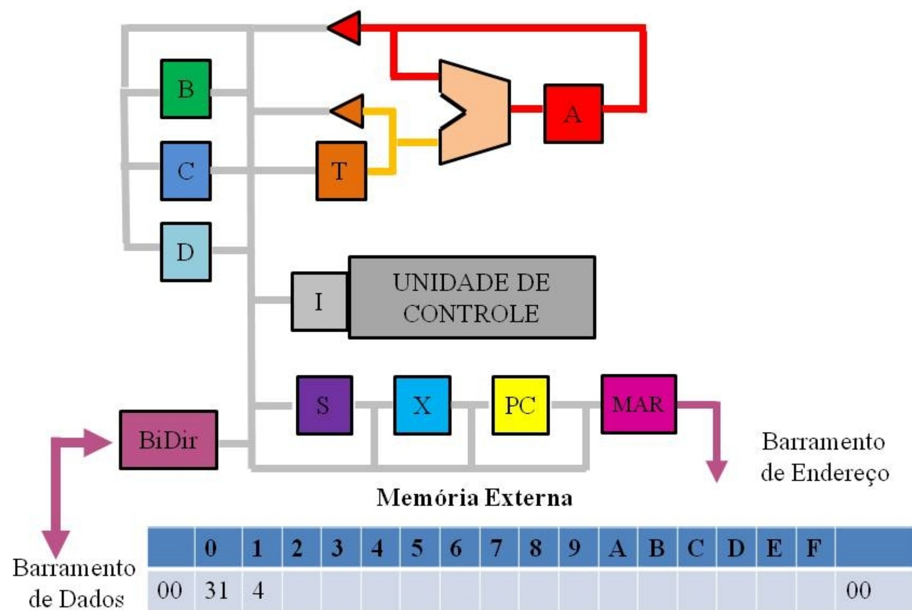


Fig. 4.1: Simulação da Arquitetura de Princeton no SimPro.

Os blocos representados por B, C, D e X compõem os registradores de uso geral que armazenam valores a serem utilizados para realizar as operações na ULA. Os registradores D e X possuem suas particularidades. Além de armazenar informações o registrador D pode funcionar como um laço de repetição (*loop*). Para tal, possui um flag, um sinal setado em '0' ou '1', cuja função é verificar se o valor da saída do registrador D é igual a '0', ou seja, ele controla a condição de parada do *loop*. O registrador X funciona como um indexador responsável por armazenar posições de vetores que podem ser usadas pelo processador.

Os blocos representados por T, A, I, S, PC e MAR são registradores específicos, ou seja, cada um é responsável por desempenhar uma função diferente no processador. O bloco T representa o registrador temporário que armazena os dados na entrada da ULA. O bloco A é o acumulador responsável por armazenar os resultados das operações da ULA. O bloco I é o registrador de instrução que armazena a instrução que será executada. O bloco S é o registrador de pilha (*Stack Pointer*). PC é o *Program Counter* que aponta para a próxima instrução a ser executada. O MAR (*Memory Address Register*) é o registrador que armazena o endereço a ser localizado na memória.

Os blocos *BiDir*, unidade de controle, memória externa e ULA representam, respectivamente: o

barramento bidirecional; a unidade de controle, que tem a função de decodificar a instrução e ativar os sinais de controle para a execução da instrução; a memória externa, que contém os dados e as instruções que o processador poderá executar; a ULA, que realiza as operações lógicas e aritméticas.

4.2.1 Exemplo de Execução de uma Instrução

Com base na arquitetura de *Princeton*, é apresentado um exemplo de simulação da execução de uma instrução utilizando o conjunto de blocos disponíveis no SIMPRO.

Primeiramente, considere a instrução 'LOADI S, 4', como sendo a primeira instrução a ser executada pelo processador. Esta instrução carrega o valor imediato 4, para o registrador S. Considere também que o valor 31 que compõe a posição, representada em hexadecimal, 0 da memória externa seja o OPCODE da instrução LOADI e que o conteúdo da posição 1 da memória seja o valor 4 que será carregado no registrador S.

No início da execução da instrução LOADI, todos os componentes estão zerados e, portanto, o valor do PC é '0'. O PC disponibiliza seu conteúdo, o valor '0', no barramento como pode ser observado na Figura 4.2.

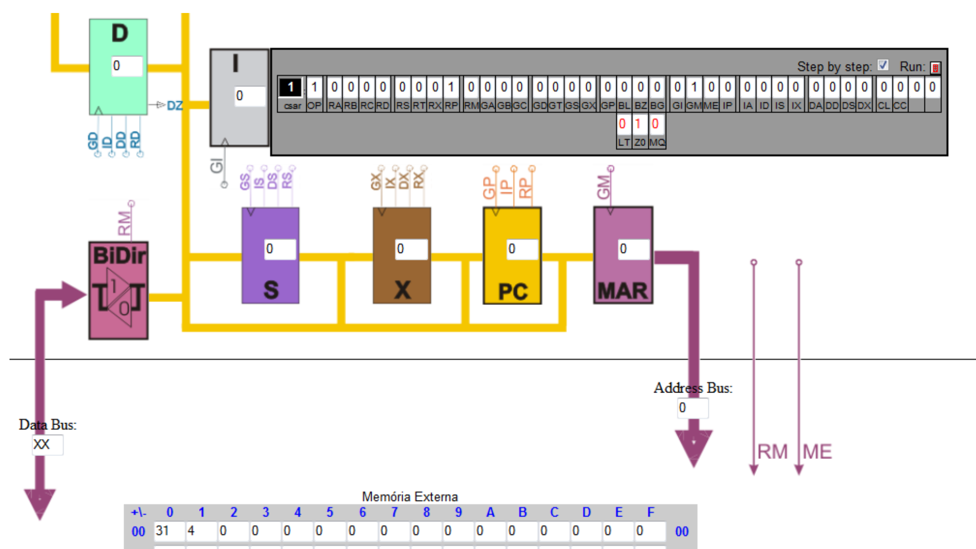


Fig. 4.2: Simulação da Instrução LOADI S - Conteúdo do PC.

O valor '0' disponibilizado pelo PC indica o endereço no MAR. O MAR faz a leitura do barramento e o endereço da posição '0', é localizado na memória externa, como é mostrado na Figura 4.3.

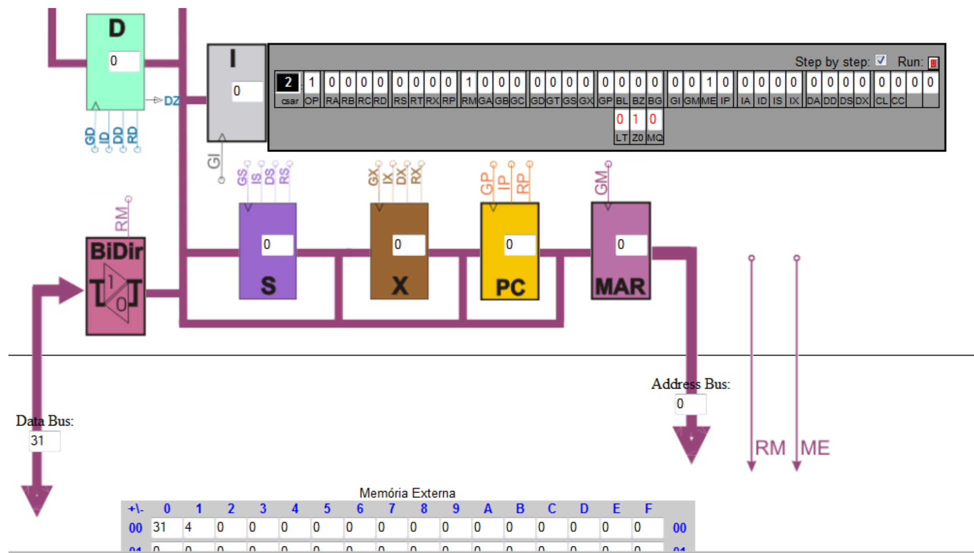


Fig. 4.3: Simulação da Instrução LOADI S - Leitura do Barramento pelo MAR.

O endereço 0 na memória, contém o valor 31. Este valor é disponibilizado no barramento e será lido pelo registrador de instruções I, como é observado na Figura 4.4.

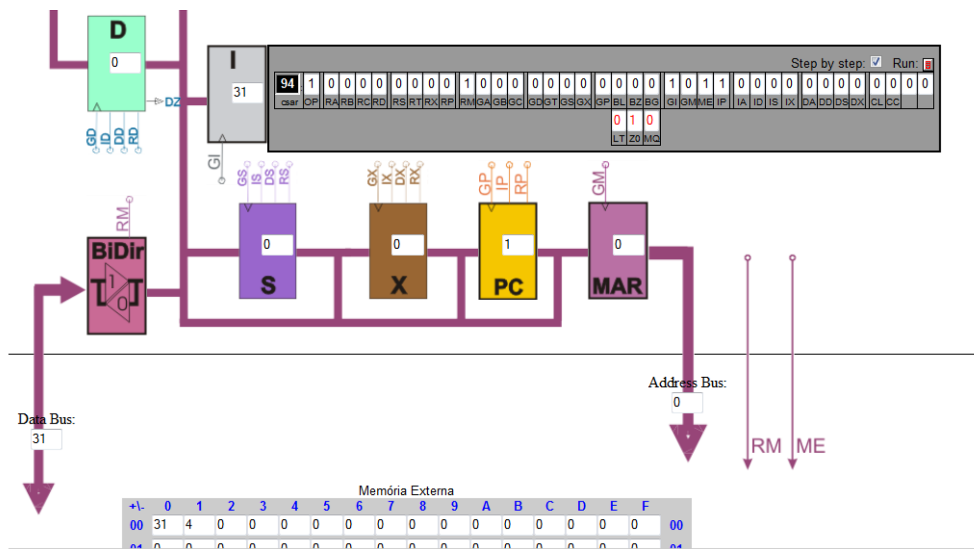


Fig. 4.4: Simulação da Instrução LOADI S - Leitura do Barramento pelo RI.

O valor do registrador I é enviado para a unidade de controle que decodificará o valor 31, interpretando que é uma instrução LOADI e ativará os sinais de controle para a execução dessa instrução.

O próximo passo, é a busca pelo operando 4, então o PC é incrementado para 1, como mostra a Figura 4.5.

Fig. 4.5: Simulação da Instrução LOADI S - Incremento do PC.

O conteúdo do PC é colocado no barramento. O MAR faz a leitura do barramento e a posição 1 é localizada na memória externa, como pode ser visualizado na Figura 4.6.

Fig. 4.6: Simulação da Instrução LOADI S - Leitura do Barramento pelo MAR.

A posição de memória 1 tem como conteúdo o valor 4 que será dispibilizado no barramento. O registrador S faz a leitura do barramento e armazena o valor 4, como é mostrado na Figura 4.7.

Fig. 4.7: Simulação da Instrução LOADI S - Leitura do Barramento pelo Registrador S.

4.3 Simulação da Arquitetura de *Harvard* no SIMPRO

Outro exemplo de simulação, utilizando os recursos gráficos do SIMPRO, é a aplicação da arquitetura de *Harvard*, como pode ser observado na Figura 4.8.

Fig. 4.8: Simulação da Arquitetura de Harvard no SimPro.

Da mesma forma que a arquitetura de *Princeton*, a arquitetura de *Harvard* é composta por um conjunto de blocos que representam os diversos tipos de componentes que a compõem. Os com-

ponentes B, C, D, T, A, ULA, S, X, PC, MAR, *BiDir* e unidade de controle, possuem a mesma funcionalidade descrita na Seção 4.2. O bloco R, é um registrador específico, cuja função é armazenar o valor das constantes que compõem as instruções. O registrador de Endereço de Memória de Programa (PMAR), do inglês *Program Memory Address Register*, possui a mesma função do MAR, só que o PMAR armazena o endereço que será localizado na memória de programa.

A arquitetura de *Harvard* é composta por duas memórias: a de dados e a de programa. A memória externa de programa tem a função de armazenar as instruções do programa. Já a memória externa de dados tem a função de armazenar as variáveis do programa. Esta arquitetura apresenta barramentos distintos, de acesso, para memória de programa e de dados. O barramento de endereço de programa só permite acesso à memória externa de programa, e o barramento de endereço de dados só permite acesso à memória externa de dados.

4.4 Simulação da Unidade MAC no SIMPRO

O MAC é uma unidade de multiplicação e acumulação, ou seja, ao realizar uma operação de multiplicação o resultado é adicionado no acumulador. A Figura 4.9 ilustra como a unidade MAC é simulada no SIMPRO.

Observe que o MAC é composto por dois blocos Z e Y que representam os registradores. A função de Z e Y é armazenar temporariamente os valores, que serão utilizados pelo multiplicador.

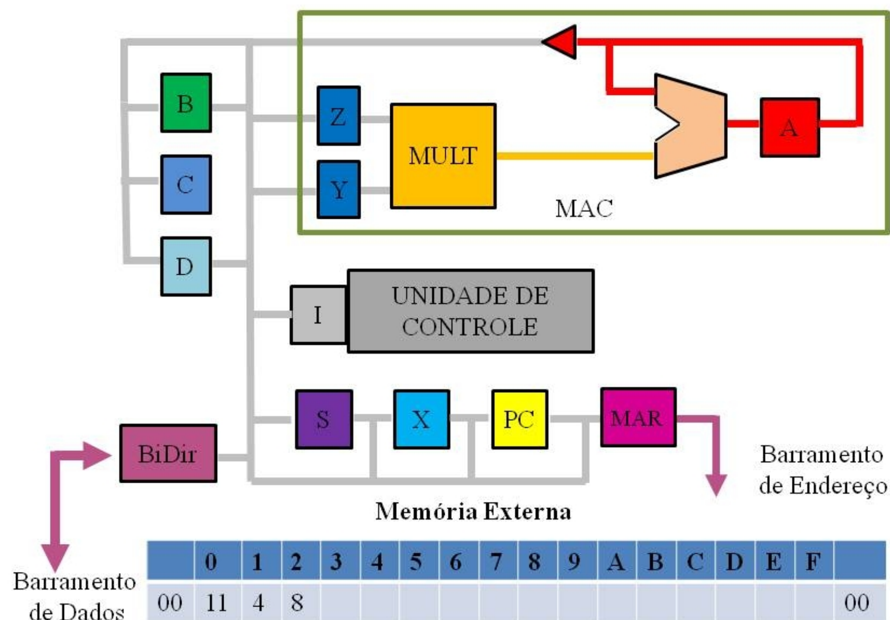


Fig. 4.9: Simulação da Estrutura MAC no SIMPRO.

O multiplicador é composto por duas entradas, relacionadas aos valores dos operandos. O resultado da multiplicação é adicionado ao acumulador.

O MAC é indicado para melhorar o desempenho de aplicações de processamento digital de sinais. Para exemplificar, considere a função 4.1 de processamento digital de sinal abaixo:

$$f(t) = a_{t_0}x_{t_0} + a_{t_1}x_{t_1} + \ldots + a_{t_n}x_{t_n} \tag{4.1}$$

Esta função é composta pelos dados *a* e *x*, que representam o coeficiente e a variável no tempo, respectivamente.

A execução desta função em um processador composto de somente uma ULA, como é o caso da arquitetura de *Princeton*, levaria dois ciclos de instrução. No primeiro, seria executada a operação de multiplicação cujo resultado seria adicionado no acumulador e, no segundo, seria realizada a operação de adição que iria somar o valor da multiplicação com o conteúdo do acumulador. Isso acontece porque somente uma única operação pode ser realizada por vez.

O MAC é mais rápido, possibilitando um desempenho melhor do processador, pois os operandos, carregados para os registradores Z e Y, são obtidos através da memória externa e, ao contrário de uma ULA comum, são carregados juntos para realizar a operação de multiplicação e, em seguida, adicionar o resultado no Acumulador.

4.5 Desenvolvimento do EMUPRO

A ferramenta QUARTUS II (versão 9.1) da Altera é utilizada para criar projetos em hardware. Ela possui uma biblioteca de componentes que envolve desde uma simples porta lógica até componentes mais complexos como memória, ULA'S e processadores, prontos para serem utilizados no projeto.

Uma vantagem desta biblioteca é que além de conter um conjunto de componentes prontos, ela permite que o projetista crie seu próprio componente através de uma ferramenta denominada *MegaWizard*, um ambiente onde o projetista poderá definir, por exemplo, o número de *bits* que irão trafegar no barramento, o tamanho das palavras, o número de entradas e de saídas, facilitando o desenvolvimento de um projeto em hardware no sentido de diminuir o tempo de projeto.

O EMUPRO fornece uma biblioteca de recursos (módulos) que pode ser usada pelos alunos para criar seus processadores. Ele foi baseado no SIMPRO e, por isso, é composto pelos mesmos módulos utizados pelo simulador. Esta biblioteca é composta, por exemplo, por um conjunto de registradores, ULA, unidade de controle e memória.

Os registradores são responsáveis por armazenar os operandos que serão utilizados pela ULA. A ULA realiza operações aritméticas de adição, subtração, divisão, multiplicação e operações lógicas. A unidade de controle é responsável por gerar todos os sinais lógicos necessários para a sequência

de microoperações que compõem uma instrução. A memória é usada para armazenar dados e/ou programas.

Assim como o SimPro, o exemplo aplicado ao EmuPro foi baseado em um processador de 8 *bits*. Contudo, diante da necessidade, o professor pode facilmente alterar esta configuração para 16 *bits*, 32 *bits* ou 64 *bits*, de acordo com a suas exigências.

4.5.1 Componente Registrador

Há dois tipos de registradores: o registrador TFF e UP/DOWN. O registrador TFF foi utilizado para representar os registradores de propósito geral, o Temporário e o registrador de endereço de memória MAR. Ele consiste de quatro entradas e uma saída, como pode ser observado na Figura 4.10.

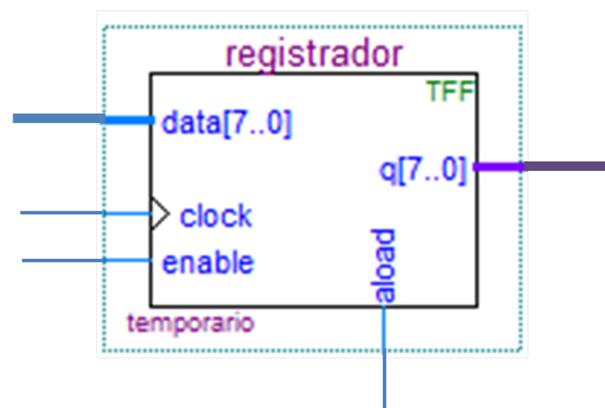


Fig. 4.10: Registrador TFF.

A primeira entrada está associada com a escrita dos dados. Ela consiste de dados de 8 *bits*, que variam de $[7 \ldots 0]$, e, quando habilitada, lê o valor do barramento e escreve no registrador. Outra entrada é o clock, ou seja, quando ocorre o pulso de clock sensível à borda de subida, as informações serão escritas ou lidas no/do registrador.

A entrada *enable* é responsável pelo controle de entrada e de saída das informações, ou escrita/leitura dos dados. A última entrada denominada *aload* permite carregar os dados de forma assíncrona, ou seja, independente do pulso de clock. Dessa forma, as informações que estão no barramento, somente serão lidas/escritas no pulso do relógio. A saída, "*q*" dos dados é composta por 8 *bits* e utilizada para realizar a leitura dos registradores.

Para os demais registradores, como por exemplo, o Acumulador, os registradores *Stack Pointer* (S) e *Program Counter* (PC) foram utilizados contadores, como pode ser observado na Figura 4.11.

Esta figura mostra o registrador *UpDown* que possui cinco entradas.

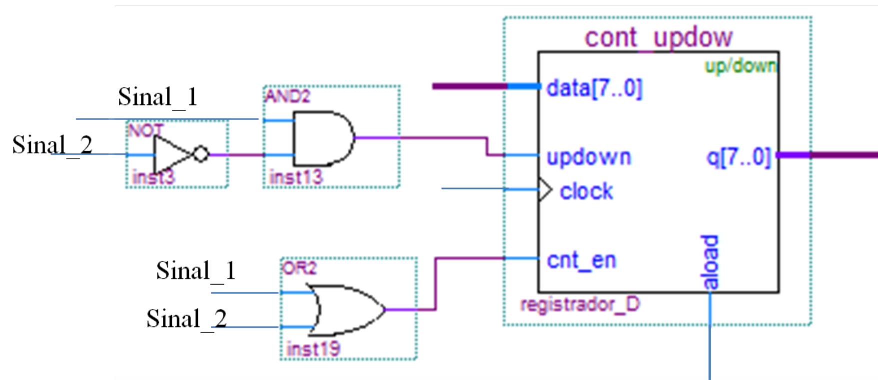


Fig. 4.11: Registrador UpDown.

A primeira entrada é a `UpDown` que tem a função de incremento ou decremento. Esta entrada está ligada a uma porta lógica AND que possui duas entradas, representadas por *Sinal_1* e *Sinal_2*, respectivamente. Portanto somente quando os dois sinais forem iguais a '1' é que o sinal de controle `UpDown` será ativado.

A entrada de *Sinal_2* da porta AND tem uma porta NOT associada. Consequentemente, a sua saída é sempre o inverso de sua entrada, isto é, se o *Sinal_2* está em nível alto, a saída da porta NOT estará em nível baixo e vice-versa.

A segunda entrada representada por `cnt_enable` tem a função de permitir o incremento ou o decremento desses registradores. Esta entrada está ligada a uma porta lógica OR que será ativada sempre que ou o *Sinal_1* ou o *Sinal_2* forem iguais a '1'.

Outro componente utilizado é um *buffer* que possui um sinal de controle, Sinal_3, que quando ativado, permite que os dados que estão no registrador sejam colocados no barramento possibilitando, assim, a leitura de dados. Este *buffer* está associado à saída dos registradores. A Figura 4.12 mostra um exemplo.

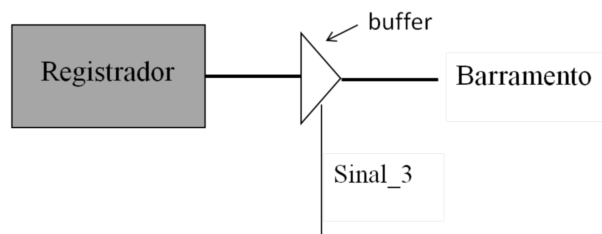


Fig. 4.12: Buffer de Leitura.

4.5.2 Componentes da Unidade Lógica e Aritmética

A ULA é constituída por cinco componentes, sendo um somador/subtrator, um divisor e um multiplicador, os quais são responsáveis pela parte aritmética. Já a parte lógica é composta por uma porta OR e outra AND.

Componente Somador/Subtrator

As operações de adição e de subtração são realizadas através do mesmo componente, como pode ser observado na Figura 4.13.

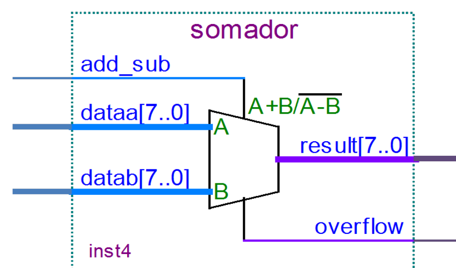


Fig. 4.13: Somador e Subtrador.

Este componente consiste de duas entradas de dados de 8 *bits* definidas como `dataa[7..0]` e `datab[7..0]` que estão associadas aos operandos. O sinal de controle `add_sub` determina se a operação vai ser de soma ou de subtração. Se for soma, `add_sub` é igual a um (1), caso contrário é zero (0). O resultado é colocado na saída `result[7..0]`. Finalmente, o sinal de controle de saída, `overflow`, indica quando a soma ou a subtração entre os operandos, ultrapassou o limite da representação númerica utilizada. Para este projeto, não se está levando em consideração este sinal, somente os sinais > (maior que), < (menor que) e 0 (zero), foram considerados, portanto, ele não foi implementado, porque não faz parte do projeto.

Componente Divisor

O componente responsável pela divisão é mostrado na Figura 4.14. O divisor é constituído por duas entradas, `numer[7..0]` e `denom[7..0]`, que representam o dividendo e o divisor, respectivamente, ambos com 8 *bits*. O resultado é obtido através da saída `quotient[7..0]`. A outra saída denominada `remain[7..0]` representa o resto da divisão, mas que não está sendo levado em conta para este projeto e por isso, não foi implementado, porque não faz parte do projeto.

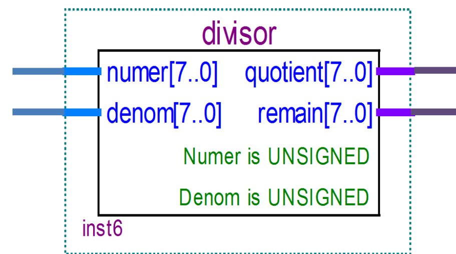


Fig. 4.14: Divisor.

Multiplicador

O multiplicador possui duas entradas de 4 *bits* cada, ou seja, somente os *bits* menos significativos são levados em consideração. Estas entradas são representadas por `dataa[3..0]` e `datab[3..0]`, e a saída `result[7..0]` de 8 *bits*, como pode ser observada na Figura 4.15.

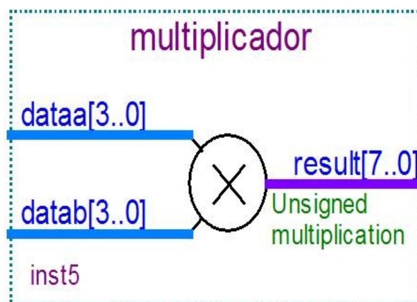


Fig. 4.15: Multiplicador.

Componentes Lógicos

As operações lógicas realizadas pela ULA no EMUPRO consistem de dois componentes AND e OR, que possuem duas entradas e uma saída, como pode ser observado na Figura 4.16.

Fig. 4.16: Portas Lógicas AND e OR.

A diferença entre estas portas é que as entradas da porta AND devem ser iguais a um (1) para que sua saída seja igual a 1. Por outro lado, para a porta OR, basta que uma de suas entradas seja igual a 1, para que sua saída seja igual a 1.

4.5.3 Multiplexador

Para que a ULA identifique qual das operações deve ser selecionada, é necessário o uso de um multiplexador como pode ser observado na Figura 4.17.

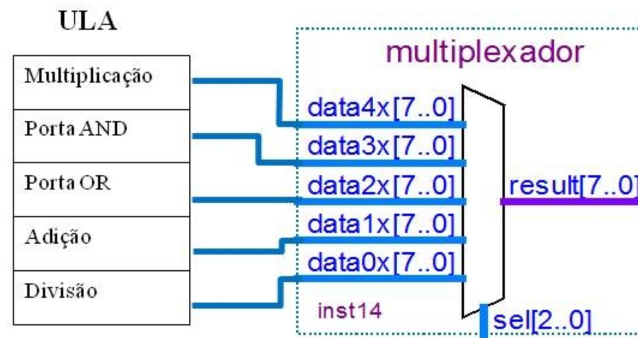


Fig. 4.17: Multiplexador.

O multiplexador é composto por *n* entradas, mas somente uma será selecionada, para ser a saída. Como o projeto desta ULA é composto por cinco componentes, o multiplexador terá cinco entradas as quais são representadas por `data0x[7..0]`, `data1x[7..0]`, `data2x[7..0]`, `data3x[7..0]` e `data4x[7..0]`, sendo que estas entradas estão associadas às operações de divisão, adição/subtração, OR, AND e multiplicação, respectivamente.

Os sinais de controle `sel[2..0]` é composto por 3 *bits*. Através destes sinais é que a operação a ser executada pela ULA, será selecionada. A saída `result[7..0]` indica a operação a ser executada.

4.5.4 Tratamento de Exceção

Uma exceção é um estado que pode mudar a sequência de execução de um programa. Este componente identifica o estado do Acumulador, como pode ser observado na Figura 4.18. Ele possui a entrada `dataa[7..0]` que receberá o resultado da operação realizada pela ULA e que está armazenado no Acumulador. Este registrador é constituído por três saídas `aeb`, `agb` e `alb`, que indicarão se o estado resultante do acumulador é igual a '0', maior que '0' ou menor que '0', respectivamente.

As três saídas do circuito de exceção do acumulador são conectadas a um conjunto de portas AND e OR, como pode ser observado na Figura 4.19, que ilustra o circuito de carregamento do PC. que juntamente com os sinais de controle BZ, BG e BL, ativam o incremento do PC.

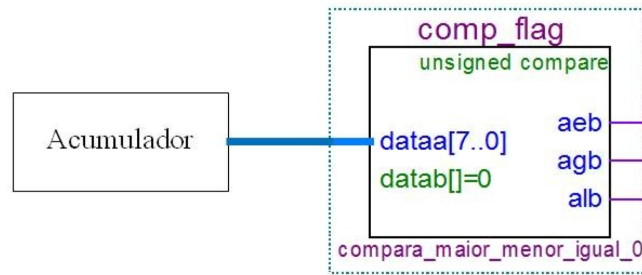


Fig. 4.18: Circuito de Exceção do Acumulador.

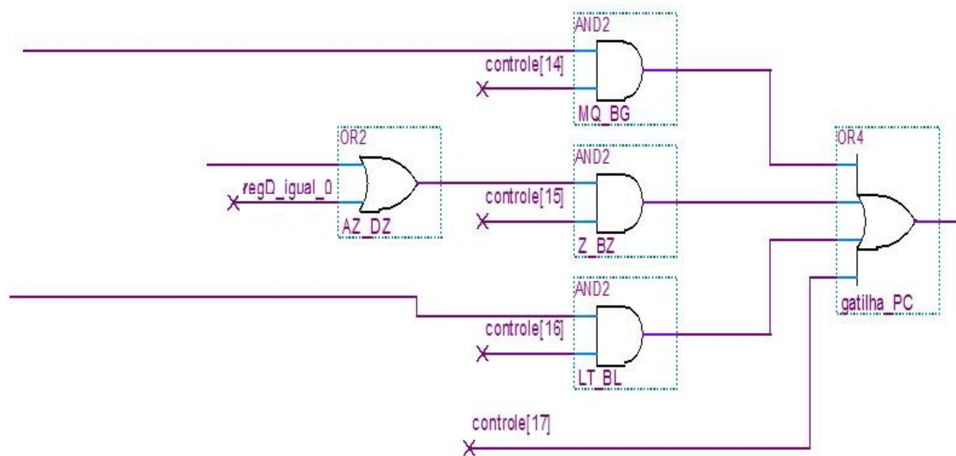


Fig. 4.19: Circuito de Carregamento do PC.

O PC pode ser carregado através de um desvio incondicional (*jump*), representado pelo sinal de controle ('controle[17]'), pelo desvio (menor que 0) representado por LT_BL ('controle[16]'), pelo desvio ('igual a 0'), sinal de controle Z_BZ ('controle[15]') e pelo desvio 'maior que 0', representado por MQ_BG ('controle[14]').

Através dessas condições de desvios é que o PC poderá ser carregado.

4.5.5 Componente Decodificador

O decodificador, mostrado na Figura 4.20, é o responsável por decodificar o OPCODE, da instrução, que chega do registrador de instruções (RI) para encontrar a linha correspondente na LUT.

O decodificador possui duas entradas sendo que uma delas contém o endereço `address[7..0]` de 8 *bits* que chega do RI, cujo conteúdo deverá ser decodificado. A outra entrada é o sinal de `clock`. A saída `q[8..0]`, de 9 *bits*, é o valor decodificado a ser encontrado na LUT.

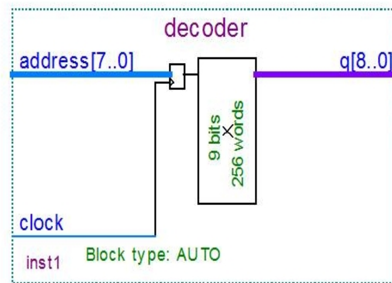


Fig. 4.20: Decodificador de Instruções.

4.5.6 Componente *Look-Up Table*

A LUT é representada por uma memória ROM, constituída de uma porta, como pode ser observado na Figura 4.21.

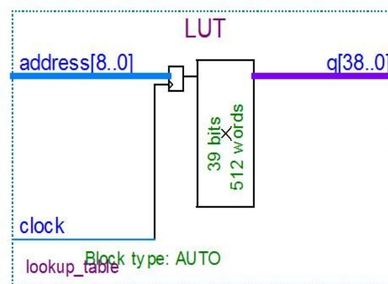


Fig. 4.21: Look-Up Table.

A LUT é um dos principais componentes que fazem parte do EMUPRO pois, além de representar a unidade de controle microprogramada, é através dela que os sinais de controle para executar uma determinada instrução serão ativados. A memória desta LUT é formada por 512 palavras de 39 *bits*, sendo que cada um destes *bits* representam um dos 39 sinais de controle do processador. Quando chega o valor decodificado, este valor é endereçado pela LUT através da entrada `address[8..0]` que irá indicar o início da execução da instrução, bem como os sinais de controle que deverão ser ativados para realizar esta instrução.

4.5.7 Componente Memória Externa

O componente da memória externa tem a função de armazenar as instruções e os dados que fazem parte do EMUPRO. Este componente é constituído por uma memória de 256 palavras de 8 *bits*, duas entradas e uma saída, como pode ser observado na Figura 4.22.

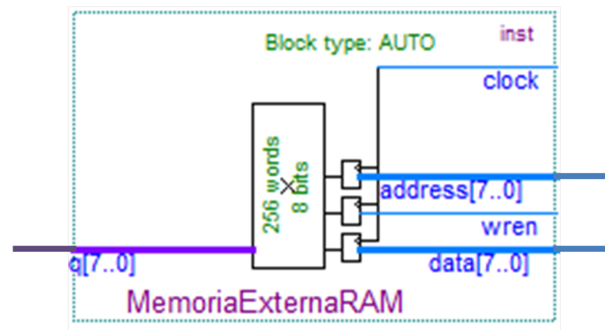


Fig. 4.22: Memória Externa.

A entrada `address[7..0]` recebe o endereço do registrador de endereço de memória (MAR), para localizar certa instrução ou dado na memória externa. A entrada `data[7..0]` tem a função de gravar os dados na memória. A saída `q[7..0]` envia as informações que representam, as instruções e os dados que serão utilizados pelo processador. O sinal de controle `wren` (*write enable*) tem a função de habilitar as operações de escrita na memória.

A saída `q[7..0]` é ligada a um *buffer* bidirecional que controla a leitura e escrita dos dados na memória, baseado em dois sinais de controle de acesso à memória: *sinal_1* relacionado a ME (*Memory Enable*) e *sinal_2*, relacionado a leitura da memória (RM). Quando for realizada uma operação de escrita ME = 1 e RM = 0, se for leitura ME = 1 e RM = 1. As portas AND ligadas ao *buffer* são usadas para controlar a leitura e a escrita dos dados na memória.

4.6 Emulação da Arquitetura de Princeton no EMUPRO

A emulação da arquitetura de Princeton foi realizada através do ambiente de desenvolvimento QUARTUS II. Este ambiente proporciona uma interface que permite ao projetista criar aplicações utilizando um conjunto de componentes que fazem parte da biblioteca do ambiente computacional. A Figura 4.23 apresenta o ambiente do QUARTUS II.

Assim como o SIMPRO, o EMUPRO possui uma estrutura modular, permitindo que novos componentes sejam adicionados ao projeto.

Fig. 4.23: Emulação da Arquitetura de *Princeton* no SIMPRO.

4.7 Resumo do capítulo

Neste capítulo, foi apresentado o desenvolvimento SIMPRO e do EMUPRO, descrevendo cada um dos componentes que os compõem, bem como descrevendo as principais características de cada um, explicitando individualmente como cada um desses componentes foram implementados.

Para o exemplo de simulação, foram utilizadas as arquiteturas de *Princeton* e *Harvard*, e a unidade MAC. Também foi apresentado um exemplo de execução de instrução na arquitetura de *Princeton*, descrevendo passo a passo o ciclo de execução, sinalizando todos os componentes que fizeram parte da execução da instrução. Para o exemplo de emulação, foi utilizada a arquitetura de *Princeton*, apresentando o ambiente de desenvolvimento do QuartusII, que fornece uma interface amigável e uma biblioteca personalizada para o desenvolvimento dos seus componentes.

No próximo capítulo, serão apresentadas as conclusões e os trabalhos futuros.

Capítulo 5

Conclusão

Neste capítulo, é apresentado o ambiente computacional proposto. Além disso, são sugeridos alguns trabalhos futuros que levam em conta a experiência adquirida.

Este trabalho apresentou o desenvolvimento do ambiente computacional (*framework*) MODPRO que consiste em uma ferramenta didática para o ensino de arquitetura de computador, objetivando mostrar como o processador e seus componentes funcionam.

O MODPRO é composto pelo simulador SIMPRO e pelo emulador EMUPRO sendo que ambos tem como função ilustrar o funcionamento do processador. Através do SIMPRO é possível realizar a simulação do processador, acompanhando todos os sinais de controle que serão ativados e o fluxo de dados para a execução de uma determinada instrução. O SIMPRO permite que a execução da instrução seja realizada passo a passo. Já o EMUPRO, permite que o aluno verifique o funcionamento da execução das instruções em tempo real através do FPGA.

Este trabalho não se compromete com o desempenho do processador que será usado no SIMPRO ou no EMUPRO. O objetivo principal tanto do SIMPRO quanto do EMUPRO é demonstrar como o processador funciona, de modo que auxilie na compreensão do seu funcionamento por parte do aluno e facilite a metodologia de ensino por parte do professor.

Assim sendo, os principais compromissos com esta ferramenta são a facilidade de ensino que o professor poderá moldar a arquitetura desejada para ser apresentada aos alunos de forma bastante simples e intuitiva. Tanto o EMUPRO quanto o SIMPRO são modulares e aberto à adição de novos recursos, permitindo o ensino de estruturas de processamento diferentes.

5.1 Trabalhos Futuros

O *framework* usado para ensinar o funcionamento de um processador simplificado foi apresentado aos alunos através do curso "Introdução à Computação Digital"da Faculdade de Engenharia Elétrica e Computação - FEEC. Primeiramente, foi mostrado aos alunos como são realizadas as operações de cada módulo tal como, por exemplo, a maneira como os registradores interagem entre si e com a ULA. Nesse sentido, o processador foi construído, um conjunto de instruções foi definida e pequenos programas foram escritos pelos alunos.

Baseado nesta experiência, novos exemplos de arquiteturas estão sendo estudados. Entre estes estudos, destaca-se a simulação e emulação de arquiteturas multicore tais como a estrutura SIMD e a estrutura MIMD [7]. Outro trabalho está relacionado ao desenvolvimento de uma interface para o SIMPRO, visto que a interface do emulador já faz parte do ambiente de desenvolvimento QUARTUS II.

A seguir será exposto como seria realizada o desenvolvimento dessas duas sugestões.

5.1.1 Exemplo de Arquitetura Paralela no ModPro

Como exemplo da implementação de estruturas de processamento paralelo no ambiente ModPro, considera-se a seguir dois exemplos, um considerando uma estrutura SIMD e outro considerando uma estrutura MIMD. Nos dois exemplos, para concluir sua implementação faz-se necessário o desenvolvimento de um módulo de gerenciamento de memória.

Considere os processadores CORE 1 e CORE 2, cada um com sua memória cache, compartilhando a mesma memória de programa e a mesma memória externa, como pode ser observado na Figura 5.1. O problema nessa estrutura está relacionado ao acesso à memória externa pelos dois processadores, ou seja, CORE 1 e CORE 2 não podem acessar simultâneamente a memória externa, pois haverá conflito de informações. Para evitar esta situação, a solução seria criar um "gerenciador de memória", um componente cuja função seja gerenciar os acessos dos processadores à memória.

Outra situação semelhante é mostrada na Figura 5.2. Novamente considere os dois processadores representados por CORE 1 e CORE 2.

Para essa estrutura, cada processador, além de possuir sua própria cache, é composto também por sua própria memória de programa. Os processadores compartilham uma memória externa. Novamente existe o problema de sincronismo em relação aos acessos a esta mémória efetuados tanto pelo processador CORE 1 quanto pelo processador CORE 2. Diante disso, a solução seria a adição de um gerenciador de memória que controlaria os acessos à memória.

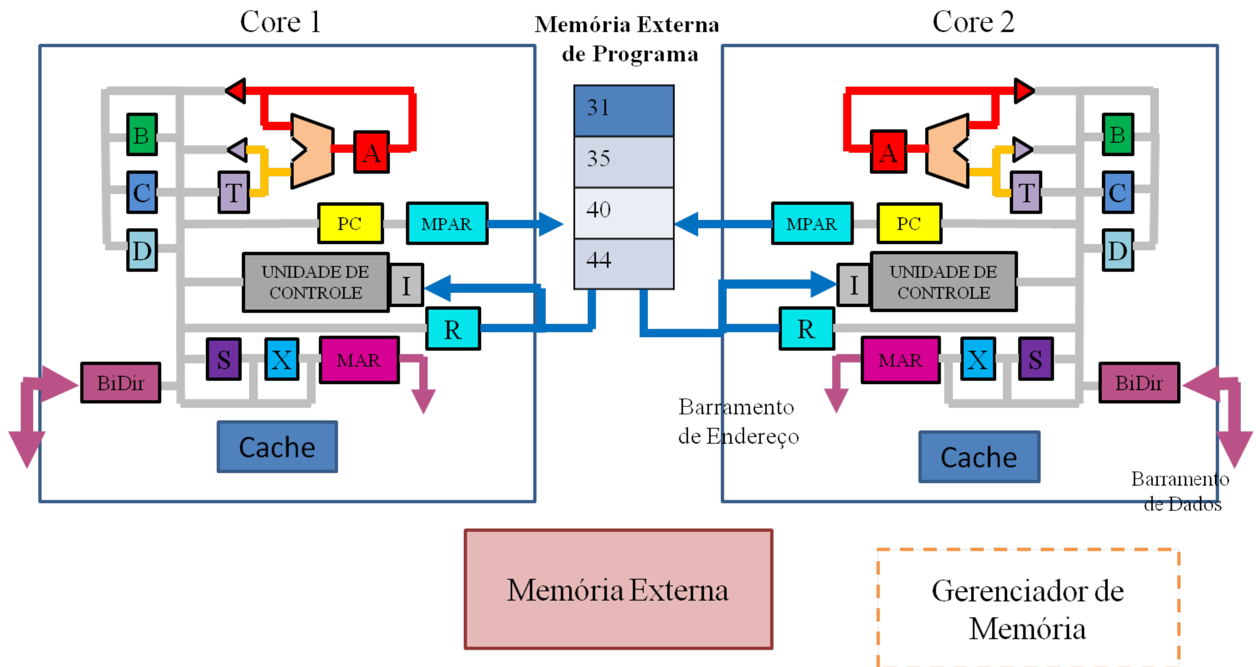


Fig. 5.1: Simulação da Estrutura SIMD.

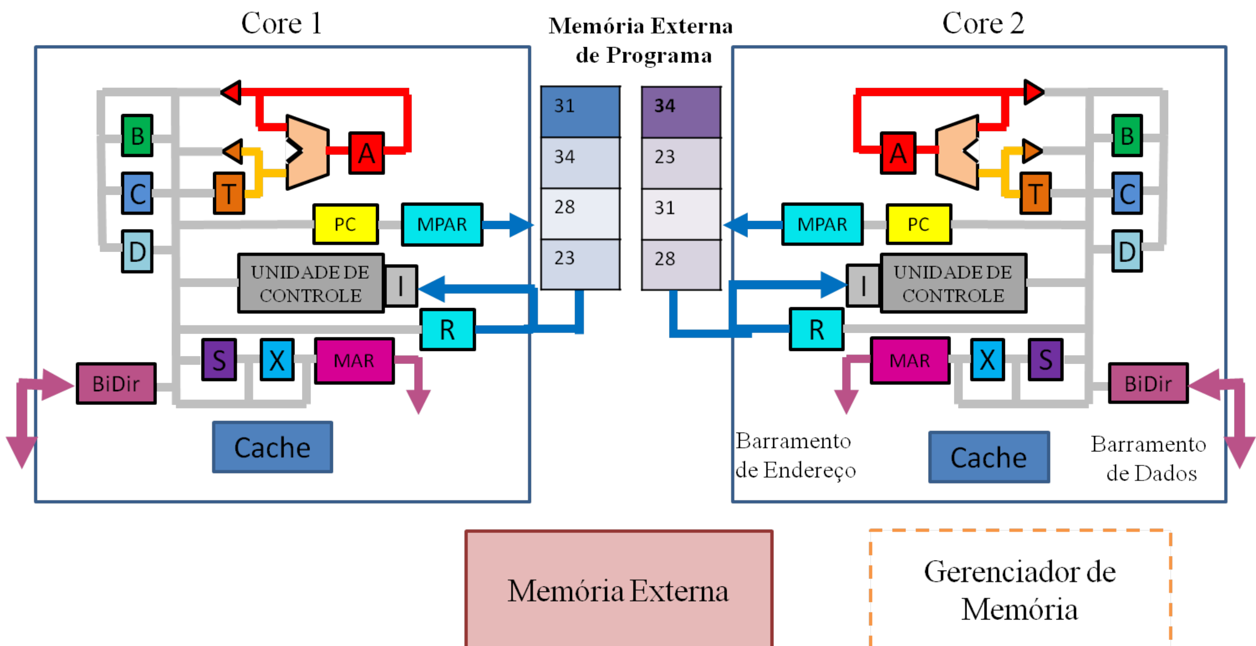


Fig. 5.2: Simulação da Estrutura MIMD.

Foi considerado que o desenvolvimento desta estrutura de processamento paralelo fugia ao escopo de um curso de introdução a processamento digital, alvo inicial de toda esta proposta. Assim sendo este estudo foi deixado para trabalho futuro.

5.1.2 Interface para o SIMPRO

O SIMPRO não possui uma interface visual para o desenvolvimento das estruturas de processamento, ao contrário do EMUPRO onde o próprio ambiente do QUARTUS II consiste numa interface mais amigável. Diante disso, é proposto desenvolver um *front-end*, uma interface para uma configuração amigável do processador. Esta interface seria composta pelos seguintes recursos:

- Blocos como: barramentos, registradores, etc.;
- Recursos de arrastar, copiar, apagar e interligar;
- Microprogramação;
- Programação.

O conjunto de blocos representa os componentes que fazem parte do processador, como registradores, ULA, barramentos. Dessa forma, possibilita ao professor moldar a arquitetura com os diferentes tipos de blocos de forma a moldar-se ao conteudo da disciplina. Esta interface permitiria, também, fazer uma microprogramação e programação, como exemplo, implementar uma LUT ou desenvolver um programa que possa ser usado em algum projeto.

O desenvolvimento desta interface amigável, não foi considerada nesta etapa da proposta do ambiente. Isto se justifica por se tratar de outro nível de desenvolvimento. Considerou-se que primeiramente seria necessário o estabelecimento de "massa crítica"de exemplos, para depois sintetizá-los numa ferramenta de descrição de módulos e de interação. Para este desenvolvimento, espera-se que recurso muito recentes para a web, como os padrões HTML5 [28] e CSS3 [29], serão de fato consolidados em todos os browsers.

Referências Bibliográficas

[1] S. Brown and J. Rose. Fpga and cpld architectures: a tutorial. *IEEE Design Test of Computers*, 13(2):42 –57, summer 1996.

[2] Andrew S. Tanenbaum. *Structured Computer Organization*. Prentice-Hall, 4a. edition edition, 1999.

[3] William Stallings. *Arquitetura e Organização de Computadores*. Pearson Education do Brasil, 8a. edição edition, 2010.

[4] Li Tan. *Digital Signal Processing Fundamentals and Aplications*. Elsevier Inc., 2008.

[5] Konstantin Solnushkin. Harvard computer architecture. Technical Report 6085/1, Universidade Politécnica do Estado de San Petersburg, 2007.

[6] X. Ke, Z. Kejia, L. Qinng, and M. Hao. The architecture comparison and the vlsi implementation of the 32bit embedded risc. In *2003 IEEE 5th International Conference on ASIC*, 2003.

[7] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C–21:948–960, Set 1972.

[8] J. L. Hennessy and D. A. Patterson. *Arquitetura de Computadores Uma Abordagem Quantitativa*. Editora Campus, 3a. edicao edition, 2003.

[9] G. Blake, R.G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Society*, 26:26–27, 2009.

[10] Bruno Cardoso, Sávio R. A. dos Santos Rosa, and Tiago M. Fernandes. Multicore. Technical report, UNICAMP, 2005.

[11] Douglas Camargo Foster. Arquiteturas multicore. Technical report, Universidade Federal de Santa Maria, 2006.

[12] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing, Principles, Algorithms and Applications*. Prentice-Hall, 3a. edition edition, 1996.

[13] Rafael Astuto Arouche Nunes, Marcelos Portes de Albuquerque, and Marcio Porter de Albuquerque. Introdução a processadores de sinais digitais. Technical Report CBPF-NT-001/2006, Centro Brasileiro de Pesquisas Físicas, Fevereiro 2006.

[14] Jennifer Eyre and Jeff Bier. Dsp processors hit the mainstream. *Computer*, 31:51–59, 1998.

[15] Paulo S. R. Diniz, Eduardo A. B. da Silva, and Sergio L. Netto. *Digital Signal Processing, System Analysis and Design*. Cambridge University Press, 2002.

[16] P. Koopman. Microcoded vs. hard-wired control. Technical report, Carnegie Mellon University, 1997.

[17] *Dicionário Michaelis*. Editora Melhoramentos.

[18] Altera. Quartus ii design software, Dec 2011.

[19] J. Banks. Introduction to simulation. In P.A. Farrington, H.B. Nembhard, D. T. Sturrock, and G.W. Evans, editors, *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 7–13, Dec 1999.

[20] I. McGregor. The relationship between simulation and emulation. In *Proceedings of the Winter Simulation Conference, 2002*, volume 2, pages 1683–1688, Dec 2002.

[21] Inc. The MathWorks. Simulink - simulation and model-based design, Dec 2011.

[22] Fabian Vargas, Juliano Benfica, and Marlon Moraes Marcelo Mallmann. *Tutorial do Simulador Spice*. Pontifícia Universidade Católica do Rio Grande do Sul, Set 2007.

[23] ARENA. Arena simulation software, Dec 2011.

[24] José Raimundo de Oliveira. Simla - simulador de micro processadores. Master's thesis, FEC–UNICAMP, 1979.

[25] B. Kiline, M. Maerz, and P. Rosenfeld. The in-circuit approach to the development of microcomputer-based products. In *Proceeding of IEEE*, volume 64, pages 937–942, 1976.

[26] Test Technology Standards Committee. Ieee standard test access port and boundary-scan architecture. Technical report, IEEE Computer Society, 2001.

[27] S. McLaughlin. Introduction to hcs08 background debug mode. Technical Report AN3335, Freescale Semiconductor, 11 2006.

[28] W3C HTML. Html5, Dezembro 20011. http://www.w3.org/TR/html5/.

[29] W3C. Cascading style sheets, dezembro 20011. http://www.w3.org/Style/CSS/.