

UNIVERSIDADE FEDERAL DE SÃO PAULO

Laboratório de Sistemas Computacionais:  
Arquitetura e Organização de Computadores

Prof. Dr. Tiago de Oliveira

Relatório:

PC1: Conjunto de Instruções e Arquitetura Base



ICT-Unifesp  
Abril de 2017

## **Sumário**

### **1. Introdução**

### **2. Objetivo**

### **3. Fundamentação Teórica**

- (a) Sistema Computacional
  - i. Organização de um Sistema Computacional
- (b) Unidade Central de Processamento
  - i. RISC e CISC
  - ii. Arquitetura MIPS
- (c) Taxonomia de Flynn
- (d) Circuitos Integrados
  - i. FPGA

### **4. Desenvolvimento do Projeto**

- (a) Desenvolvimento do Conjunto de Instruções
- (b) Desenvolvimento do esquemático da Arquitetura Base

### **5. Referências**

# 1 Introdução

Na era das máquinas, o computador (ou os sistemas computacionais em um geral) está presente no dia-dia da sociedade, fazendo falta caso se passe um dia distante do dono. A **CPU** (*Central Processor Unit*, em português, Unidade Central de Processamento) está por trás de todas as operações que os sistemas são submetidos, sendo representado até como um cérebro em algumas interpretações. É ela que captura os dados, processa-os e imprime o que foi pedido pelo usuário. No primeiro momento, foi escolhida o conjunto de instruções que o processador irá suportar. Também foi montada sua arquitetura base, inspirada na arquitetura MIPS monociclo. Futuramente, será projetada a **Unidade de Processamento** e **Unidade de Controle**. Nomeada de **NorMIPS**, sua implementação será toda feita em **Verilog**.

## 2 Objetivo

Elaboração de um conjunto de instruções e uma arquitetura base para desenvolvimento de um **processador**. Este projeto de processador deverá funcionar em um circuito FPGA, produzindo as operações básicas aritméticas e lógicas, ações de *jumps* e leitura e escrita de dados.

Para o **Ponto de Checagem 1**, apenas deve-se escolher as instruções que comporão o Conjunto de Instrução, seus respectivos formatos de instrução e a arquitetura base, por qual funcionará o Caminho de Dados (*Datapath*).

## 3 Fundamentação Teórica

### 3.1 Sistema Computacional

**Sistema Computacional** é um conjunto de elementos eletrônicos (*hardware* que pode processar informações (dados) em forma de programas (*software*). Geralmente produzido para suporte ou automação de trabalhos e tarefas.



Figura 1: MacBook Pro 2016, computador de última geração da empresa Apple

#### 3.1.1 Organização de um Sistema Computacional

O *hardware* dos sistemas computacionais executam as mesmas operações elementares: inserção de dados, processamento, armazenamento e impressão.

Um computador é composto de dispositivos de entrada (como o teclado e *mouse*, dispositivos de saída (como o monitor), memória e processador (que faz contém o caminho de dados e a unidade de controle). O funcionamento se consiste basicamente em: os dados que chegam pelos dispositivos de entrada, são armazenados na memória para serem processados no processador e, finalmente, são impressos no dispositivo de saída.

### 3.2 Unidade Central de Processamento

**Processadores** são os componentes cruciais dos sistemas computacionais. Funcionando como o "cérebro" do computador, é ele que realiza as instruções de um programa, como operações de aritmética, lógica, entrada e saída de dados. Basicamente, sua estrutura é formada por três itens: a **Unidade Lógica Aritmética** (ou ULA), responsável por efetuar as operações lógicas e aritméticas; a **Unidade de Controle**, responsável por controlar os pontos de execução e desvios; e os **Registradores**, que armazenam os dados de processamento.



Figura 2: Esquemático da organização dos computadores

### 3.2.1 RISC e CISC

**RISC e CISC** são tipos de arquiteturas de processadores, tomando uma abordagem de projeto do conjunto de instruções. RISC, ou *Reduced Instruction Set Computer* (Computador de Conjunto de Instruções Reduzidos), é uma arquitetura que visa o menor conjunto de instruções possível. Sua implementação (a nível de *hardware*) tende a ser feita de maneira simples, dado que é pouco o número de instruções. Já a arquitetura CISC, ou *Complex Instruction Set Computer* (Computador de Conjunto de Instruções Complexo) possuem um conjunto de instruções maior. Dessa forma, sua organização se constrói de maneira complexa. Porém, para o programador, o desenvolvimento do software é mais simples, já que o grande número de instruções abrange vastas funcionalidades.



Figura 3: Intel Xeon 5500 da família de arquitetura x86. Arquiteturas x86 seguem o padrão CISC

### 3.2.2 Arquitetura MIPS

**MIPS** (*Microprocessor without Interlocked Pipeline Stages*) é uma arquitetura RISC elaborada pela **MIPS Computer Systems**. Possui um conjunto de instruções contendo 39 instruções reais e 8 pseudo-instruções (instruções que podem ser traduzidos em instruções reais-múltiplas). A arquitetura foi desenvolvida por uma equipe da Universidade de *Berkeley* conduzido por David Patterson e Carlos Séquin. MIPS dispõe de diversos tipos de aplicações, tendo como principais delas servir como um sistema computacional de apoio para projetos de processadores em "*Stage-Work*".

## 3.3 Taxonomia de Flynn

A **Taxonomia de Flynn** é um método para classificação de arquitetura de computadores a partir de como eles operam, em relação ao fluxo de instruções e fluxo de dados. 4 classes são incorporadas à Taxonomia de Flynn:

- **SISD:** *Single Instruction Single Data*, são as arquiteturas que para uma instrução, operam sobre um único dado. Exemplo: Arquitetura MIPS.

- **SIMD:** *Single Instruction Multiple Data*, são as arquiteturas que para uma instrução, operam sobre múltiplos dados. Exemplo: Computadores Vetoriais.
- **MISD:** *Multiple Instruction Single Data*, são as arquiteturas que para múltiplas instruções, operam sobre um único dado. Exemplo: Arquitetura *Pipeline*.
- **MIMD:** *Multiple Instruction Multiple Data*, são as arquiteturas que para múltiplas instruções, operam sobre múltiplos dados.

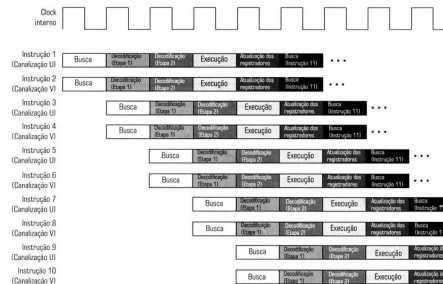


Figura 4: Esquemático de funcionamento do *Pipeline* em um processador Intel Pentium MMX

### 3.4 Circuitos Integrados

Circuitos integrados são circuitos eletrônicos que reúne diversas peças, como transistores, diodos, resistores e capacitores. Estes elementos são introduzidos em uma lâmina de silício. Essa lâmina ("chip") é montado em um bloco, de plástico ou cerâmica, conectando os terminais dos seus componentes por fios condutores.

#### 3.4.1 FPGA

FPGA ("*Field Programmable Gate Array*", em português "Arranjo de Portas Programável em Campo") é um tipo de circuito integrado projetado para configuração própria pelo usuário, seja ele um simples consumidor ou um projetista. Funciona como um dispositivo lógico programável que suporta implantação de circuitos digitais. A partir do *software* Quartus é gerado um arquivo binário com a configuração da FPGA. Esse arquivo binário contém as informações necessárias para especificar a função de cada unidade lógica e da seleção de *switches* e botões a serem utilizado no projeto. Para desenvolvimento do trabalho, fora utilizado o FPGA modelo EP4CE115F29C7, da Altera.

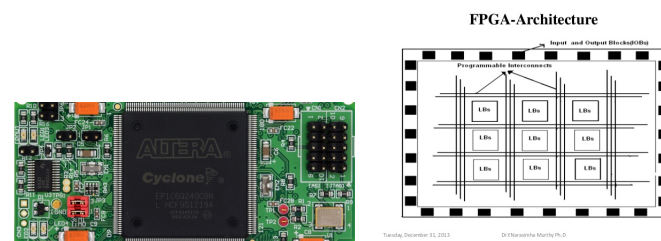


Figura 5: FPGA e o esqueleto de sua arquitetura

## 4 Desenvolvimento do Projeto

O projeto proposto consiste em desenvolver um processador que funcione em um FPGA. O circuito servirá como uma interface entre o usuário e a CPU, fazendo com que os dados sejam introduzidos e impressos em seus componentes. Além disso, o usuário poderá escolher as funcionalidades,

baseado no conjunto de instruções de *NorMIPS*.

O desenvolvimento projeto até o **Ponto de Checagem 1** foi dividido em duas etapas:

- Desenvolvimento do Conjunto de Instruções;
- Desenvolvimento do esquemático da Arquitetura Base;

#### 4.1 Desenvolvimento do Conjunto de Instruções

O processador a ser desenvolvido será baseado na arquitetura MIPS, que segue o tipo de arquitetura RISC monociclo. Devido a este fato, a montagem das instruções também foram baseadas no conjunto de instruções MIPS.

Categoria	Nome	Sintaxe da instrução	Significado	Formato
Aritmética	Add	add \$1,\$2,\$3	\$1 = \$2 + \$3 (signed)	R 0
	Add unsigned	addu \$1,\$2,\$3	\$1 = \$2 + \$3 (unsigned)	R 0
	Subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3 (signed)	R 0
	Subtract unsigned	subu \$1,\$2,\$3	\$1 = \$2 - \$3 (unsigned)	R 0
	Add immediate	addi \$1,\$2,CONST	\$1 = \$2 + CONST (signed)	I 816
	Add immediate unsigned	addiu \$1,\$2,CONST	\$1 = \$2 + CONST (unsigned)	I 916
	Multiply	mult \$1,\$2	LO = (\$1 * \$2) << 32; HI = (\$1 * \$2) >> 32;	R 0
	Divide	div \$1, \$2	LO = \$1 / \$2 HI = \$1 % \$2	R
Transferência de Dados	Load word	lw \$1,CONST(\$2)	\$1 = Memory(\$2 + CONST)	I 23
	Load halfword	lh \$1,CONST(\$2)	\$1 = Memory(\$2 + CONST) (signed)	I 25
	Load halfword unsigned	lhu \$1,CONST(\$2)	\$1 = Memory(\$2 + CONST) (unsigned)	I
	Load byte	lb \$1,CONST(\$2)	\$1 = Memory(\$2 + CONST) (signed)	I
	Load byte unsigned	lbu \$1,CONST(\$2)	\$1 = Memory(\$2 + CONST) (unsigned)	I
	Store word	sw \$1,CONST(\$2)	Memory(\$2 + CONST) = \$1	I
	Store half	sh \$1,CONST(\$2)	Memory(\$2 + CONST) = \$1	I
	Store byte	sb \$1,CONST(\$2)	Memory(\$2 + CONST) = \$1	I
	Load upper immediate	lui \$1,CONST	\$1 = CONST << 16	I
	Move from high	mfhi \$1	\$1 = HI	R
	Move from low	mflo \$1	\$1 = LO	R 0
	Move from Control Register	mfcZ \$1, \$2	\$1 = Coprocessor[Z].ControlRegister[\$2]	R
	Move to Control Register	mtcZ \$1, \$2	Coprocessor[Z].ControlRegister[\$2] = \$1	R
	Load word coprocessor	lwcZ \$1,CONST(\$2)	Coprocessor[Z].DataRegister[\$1] = Memory(\$2 + CONST)	I
	Store word coprocessor	swcZ \$1,CONST(\$2)	Memory(\$2 + CONST) = Coprocessor[Z].DataRegister[\$1]	I
Lógico	And	and \$1,\$2,\$3	\$1 = \$2 & \$3	R
	And immediate	andi \$1,\$2,CONST	\$1 = \$2 & CONST	I
	Or	or \$1,\$2,\$3	\$1 = \$2   \$3	R
	Or immediate	ori \$1,\$2,CONST	\$1 = \$2   CONST	I
	Exclusive or	xor \$1,\$2,\$3	\$1 = \$2 ^ \$3	R
	Nor	nor \$1,\$2,\$3	\$1 = ~( \$2   \$3)	R
	Set on less than	slt \$1,\$2,\$3	\$1 = (\$2 < \$3)	R
	Set on less than immediate	slti \$1,\$2,CONST	\$1 = (\$2 < CONST)	I
Deslocamento bit a bit	Shift left logical	sll \$1,\$2,CONST	\$1 = \$2 << CONST	R
	Shift right logical	srl \$1,\$2,CONST	\$1 = \$2 >> CONST	R
	Shift right arithmetic	sra \$1,\$2,CONST	$\$1 = \$2 >> CONST + \left( \left( \sum_{n=1}^{CONST} 2^{31-n} \right) \cdot \$2 >> 31 \right)$	R
desvio condicional	Branch on equal	beq \$1,\$2,CONST	if (\$1 == \$2) go to PC+4+CONST	I
	Branch on not equal	bne \$1,\$2,CONST	if (\$1 != \$2) go to PC+4+CONST	I
salto incondicional	Jump	CONST	goto address CONST	J
	Jump register	r \$1	goto address \$1	R
	Jump and link	al CONST	\$31 = PC + 4; goto CONST	J

Figura 6: Tabela com as 39 instruções da arquitetura MIPS

Porém, como visto na tabela, o Conjunto de Instruções MIPS contém 39 instruções, das quais muitas delas não serão utilizadas no *NorMIPS*. Portanto, instruções como as "*unsigned*" e "*immediate*" não foram elegidas para compor o conjunto de instruções do processador a ser projetado. Após as escolhas/filtro, o conjunto de instruções completo de *NorMIPS* contém 21 instruções, fora duas novas instruções que serão implementadas para *Input* e *Output* de dados.

NorMIPS		
R-type	I-type	J-type
add	lw	j
sub	sw	jal
mult	slli	
div	beq	
mthi	bne	
mflo		
and		
or		
xor		
nor		
sll		
sll		
srl		
jr		

Figura 7: Tabela com as 21 instruções de *NorMIPS*

### Formato de Instruções

Seguindo a linha de pensamento da arquitetura MIPS, o projeto de *NorMIPS* contém 3 formatos de instruções: **Tipo-R**, **Tipo-I** e **Tipo-J**.

As instruções pertencentes ao Tipo-R são as que operam funções lógico-aritméticas básicas, como *add*, *and* e *sll*. Já as pertencentes ao Tipo-I são as instruções que operam transferência de dados e *branches*. *Lw*, *sw* e *beq* fazem parte deste grupo. As instruções Tipo-J as funções *jump* e nela temos apenas duas instruções: *j* e *jal*.

NorMIPS						
Formato	op	rs	rt	rd	shamt	funct
Tipo R	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Formato	op	rs	rt	endereço		
Tipo I	6 bits	5 bits	5 bits	16 bits		
Formato	op	endereço				
Tipo J	6 bits	26 bits				

Figura 8: Estruturas dos formatos de instruções R,I e J

Como no MIPS, os formatos de instruções de *NorMIPS* têm 32 bits de largura e são organizadas como mostrados acima na Figura 9.

Instruções do Tipo-R são organizadas por 6 bits de **opCode** (*Operation Code*), que servirá para identificar as instruções; 15 bits dos números de registradores (**rs**, **rt** e **rd**, com 5 bits cada um); 5 bits de **shamt** (*Shift Amount*), que indica a quantidade deslocamento; e 6 bits de **funct**, porém, este campo da instrução não será utilizado e será preenchida por valores 0.

Instruções do Tipo-I são organizadas por, também, 6 bits de **opCode**; 10 bits dos números de registradores (**rs** e **rt**, com 5 bits cada um); e 16 bits de **endereço**, que indicará o número de endereço de algum registrador.

Instruções do Tipo-J são organizadas por 6 bits de **opCode** e 26 bits de **endereço**.

## 4.2 Desenvolvimento do esquemático da Arquitetura Base

Como *NorMIPS* é baseado na arquitetura MIPS monociclo, temos que será (sobre o ponto de vista do fluxo de instruções e fluxo de dados) uma arquitetura SISD. Constituído de uma Unidade de Controle, *Program Counter*, Memória de Instrução, Banco de Registradores, Unidade Lógica Aritmética, Entrada/Saída e Memória de Dados, o esquemático de *NorMIPS* pode ser visto na Figura 10 abaixo.

A **Memória de Instrução** é responsável por armazenar as instruções. Tem como entrada os 6 bits do endereço da instrução a ser executada e sua saída são os 32 bits da instrução. O **PC** (*Program Counter*, ou Contador de Programa) é encarregado de determinar a próxima instrução a ser executada. O **Banco de Registradores** contém os 32 registradores presentes do processador (*NorMIPS* trabalhará com Registradores-Registradores devido seguir a arquitetura RISC, caso fosse CISC, o ideal seria trabalhar com Registrador-Memória) que armazenarão as informações a serem processadas, suas entradas são os 5 bits dos números de registradores, são os dados a serem trabalhados. A **Unidade Lógica Aritmética** é responsável por fazer as operações aritméticas básicas, como soma e subtração, e calcular os resultados das operações de "*branches*". A **Entrada/Saída** é responsável por fazer o intermédio de receber e ceder as informações com o

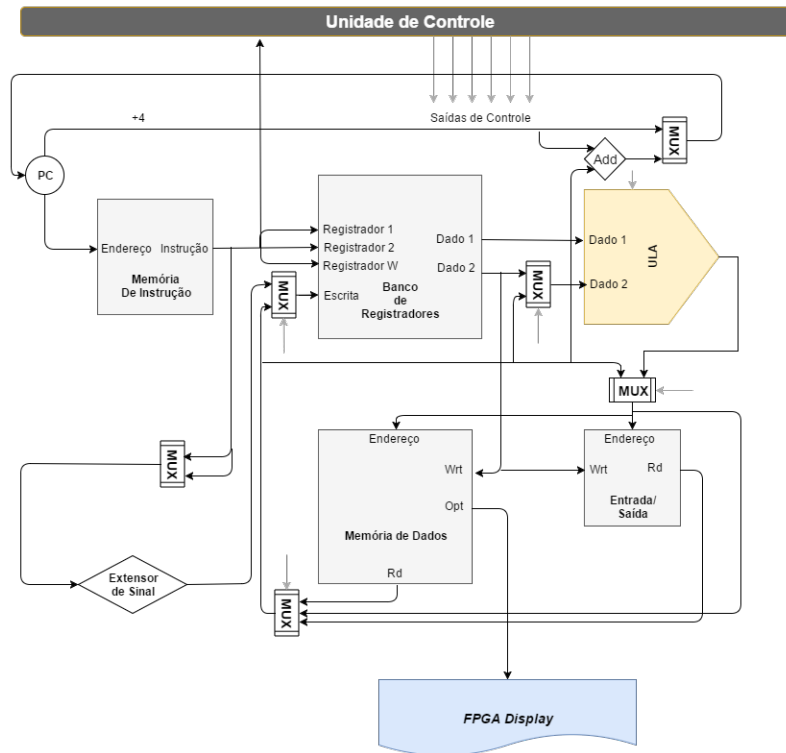


Figura 9: Esquemático da arquitetura base de *NorMIPS*

usuário. A **Memória de Dados** é responsável por armazenar as informações que acabaram de ser recebidas pelo usuário e as que já terminaram de ser processadas. A **Unidade de Controle** é responsável por determinar como cada componente irá funcionar para cada instrução. Suas saídas são ligadas aos multiplexadores, para determinar qual saída cada um deve mostrar, e na ULA, para dizer qual operação aritmética o elemento deve executar.



## Referências

[1] PATTERSON, David. *Organização e Projeto de Computadores: A Interface Hardware/Software*. Elsevier Editora, 2005;

[2] ALTERA, *Recommended HDL Coding Styles - e-book*, 2011;

[3] FLETCHER, Chris. *Verilog: always @ Blocks*. Disponível em: <<https://class.ee.washington.edu/371/peckol/doc/A>

[4] PEREIRA, Rodrigo. *Processadores Programáveis*. Disponível em: <<https://www.embarcados.com.br/processadores-programaveis-como-projetar-um-processador-em-verilog-codificacao-3/>>