Universidade Estadual do Sudoeste da Bahia – UESB
Departamento de Ciências Exatas e Tecnológicas – DCET
Colegiado do Curso de Ciências da Computação – CCComp
Circuitos Digitais

DOCUMENTAÇÃO DE UM PROCESSADOR BASEADO NO MIPS EM VHDL PARA UMA PLACA FPGA

Vitória da Conquista - BA
maio de 2018
Cássio Meira Silva
Dionleno Silva de Oliveira

DOCUMENTAÇÃO DE UM PROCESSADOR BASEADO NO MIPS EM VHDL PARA UMA PLACA FPGA

Trabalho para avaliação da disciplina Arquitetura de Computadores I, do curso de Ciências da Computação, Universidade Estadual do Sudoeste da Bahia – UESB, ministrada pelo profª Marco Antonio Dantas Ramos, período 2017.2

Vitória da Conquista - BA
maio de 2018

Esta documentação descreve o **funcionamento** do projeto de um processador de 32 bits, com módulos codificados em VHDL e esquemáticos para uma placa Altera FPGA Cyclone II modelo EP2C20F84C7.

ARQUITETURA DO CONJUNTO DE INSTRUÇÕES

A Arquitetura do Conjunto de Instruções define os tipos de instruções que serão executadas por um processador, assim como o formato de cada instrução, quantidade de bits, a forma como essas instruções acessarão registradores e memórias (modos de endereçamento), a forma como conversarão com outras unidades funcionais, etc.

Formato Tipo R

opcode	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
código da operação	registrador fonte	registrador fonte	registrador destino	deslocamento	sub código da operação

exemplo:

instrução	exemplo	
ADD registrador destino, registrador fonte, registrador fonte	ADD $t0, $s0, $s1	$t0 = $s0 + $s1
SUB registrador destino, registrador fonte, registrador fonte	SUB $t1, $s3, $s4	$t1 = $s3 - $s4

Formato Tipo I

opcode	**rs**	**rt**	**endereço**
6 bits	5 bits	5 bits	16 bits
código da operação	registrador destino	registrador fonte	endereço de memória

exemplo:

instrução	exemplo	
LW registrador destino, valor (registrador fonte)	LW $t0, 20 ($s0)	$t0 = memória [20 + $s0]

Formato Tipo Beq

Beq significa Branch On Equal,(desvie se for igual), ou seja um desvio condicional. Essa instrução força um desvio para o comando com o LABEL(nome de desvio) se o valor no registrador 1 for igual ao valor registrador 2, portanto, é uma instrução de comparação.

opcode	**rs**	**rt**	**endereço**
6 bits	5 bits	5 bits	16 bits

exemplo:

instrução	exemplo	
BEQ registrador1, registrador2, endereço de desvio	BEQ $s0, $s1, L2	#desvía para L2 se x = y

Formato Tipo Jump

As instruções J (desvio incondicional) têm apenas dois campos, sendo o OPCODE com 6 bits e o Endereço com 26 bits.

opcode	**endereço**
6 bits	26 bits

exemplo:

instrução	exemplo	
Jump	j 2500	Desvia para o endereço de destino

UNIDADES FUNCIONAIS

O projeto foi projetado de forma modularizada, com várias unidades funcionais, abaixo terá uma descrição de cada unidade:

PROGRAM COUNTER

O program counter é uma registrador de 32 bits que se atualizado a cada ciclo de instrução, ele indica qual é a posição atual na sequência de execução de um processo. Na nossa arquitetura, ele armazena o endereço da instrução sendo executada. Inicialmente ele é iniciado com zero, para executar a instrução armazenada na posição zero, no primeiro endereço da memória de instrução.

MEMÓRIA DE INSTRUÇÃO

a memória de instrução é uma unidade de memória que guarda e fornece instruções a partir de um endereço.

REGISTRADOR DE INSTRUÇÃO

é um registrador de 32 bits que guarda o código da instrução que está sendo executado.

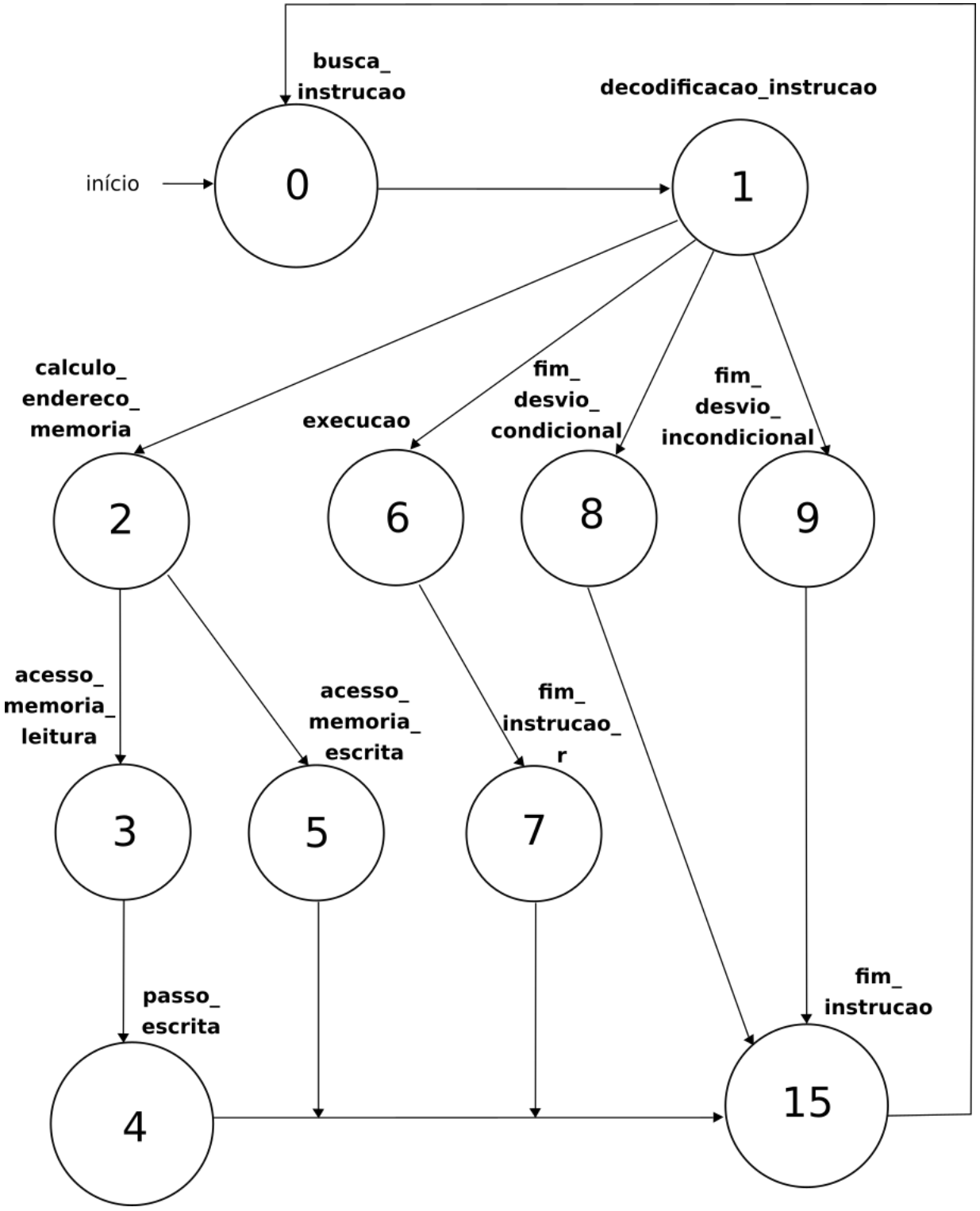
SOMADOR

é uma unidade totalmente combinacional, que incrementa um no valor do program counter a cada ciclo de instrução, de maneira a compor o endereço da próxima instrução.

UNIDADE DE CONTROLE

a unidade de controle como o próprio nome já diz, é a unidade que controla o processador. Através do opcode(código da operação), que são os bits [31..26] do código da instrução, a unidade saberá de qual tipo é determinada instrução, com isso ela era distribuir sinais com uma máquina de estado da seguinte forma:

- três sinais de 1 bit usados para controlar os multiplexadores(RegDst, UALFonte, e MemParaReg);
- três sinais para controlar as leituras e as escritas no banco de registradores e na memória de dados(EscReg, LerMem, EscMem);
- um sinal de 1 bit usado para a existência ou não de desvio condicional(DvC);
- um sinal de 1 bit usado para a existência ou não de desvio incondicional(DvI);
- um sinal de 1 bit usado para a para habilitar escrita de pc e do registrador de instrução (enable_Instrucao);
- um sinal de controle de 2 bits para a UAL(UALOp).


busca_
instrucao
decodificacao_instrucao
início
0
1
calculo_
endereco_
memoria
execucao
fim_
desvio_
condicional
fim_
desvio_
incondicional
2
6
8
9
acesso_
memoria_
leitura
acesso_
memoria_
escrita
fim_
instrucao_
r
3
5
7
passo_
escrita
fim_
instrucao
4
15

SINAIS DISTRIBUÍDOS PELAS INSTRUÇÕES EM CADA ESTADO

Instrução Tipo R

estado/sinais	enable_ instrucao	enable PC	Reg Dst	DVI	DVC	Ler Mem	Mem Para Reg	ULA Op	Esc Mem	ULA Fonte	Esc Reg
busca_ instrucao	0	0	0	0	0	0	0	00	0	0	0
decodificacao_ Instrucao	0	0	0	0	0	0	0	00	0	0	0
execucao	0	0	**1**	0	0	0	0	**10**	0	0	0
fim_instrucao_r	0	**1**	**1**	0	0	0	**0**	**10**	0	**0**	**1**
fim_instrucao	**1**	0	0	0	0	0	0	00	0	0	0

Instrução BEQ

estado/sinais	enable_ instrucao	enable PC	Reg Dst	DVI	DVC	Ler Mem	Mem Para Reg	ULA Op	Esc Mem	ULA Fonte	Esc Reg
busca_ instrucao	0	0	X	0	0	0	X	00	0	0	0
decodificacao_ Instrucao	0	0	X	0	**1**	0	X	**01**	0	0	0
fim_desvio_ condicional	0	**1**	X	0	**1**	0	X	**01**	0	0	0
fim_instrucao	**1**	0	0	0	0	0	0	00	0	0	0

Instrução SW

estado/sinais	enable_ instrucao	enable PC	Reg Dst	DVI	DVC	Ler Mem	Mem Para Reg	ULA Op	Esc Mem	ULA Fonte	Esc Reg
busca_ instrucao	0	0	X	0	0	0	X	00	0	0	0
decodificacao _Instrucao	0	0	X	0	0	0	X	XX	0	0	0
calculo_ endereco_ memoria	0	0	X	0	0	0	X	**00**	0	**1**	0
acesso_ memoria_ escrita	0	**1**	X	0	0	0	X	00	**1**	**0**	0
fim_instrucao	**1**	**0**	**0**	0	0	0	**0**	00	**0**	0	0

Instrução LW

estado/sinais	enable_ instrucao	enable PC	Reg Dst	DVI	DVC	Ler Mem	Mem Para Reg	ULA Op	Esc Mem	ULA Fonte	Esc Reg
busca_ instrucao	0	0	0	0	0	0	0	00	0	0	0
decodificacao _Instrucao	0	0	0	0	0	0	0	00	0	0	0
calculo_ endereco_ memoria	0	0	0	0	0	**1**	0	**00**	0	**1**	0
acesso_ memoria_ leitura	0	0	0	0	0	**1**	**1**	**00**	0	**1**	0
passo_escrita	0	**1**	0	0	0	**1**	**1**	**00**	0	**1**	**1**
fim_instrucao	**1**	0	0	0	0	0	0	00	0	0	0

Instrução JUMP

estado/sinais	enable_ instrucao	enable PC	Reg Dst	DVI	DVC	Ler Mem	Mem Para Reg	ULA Op	Esc Mem	ULA Font e	Esc Reg
busca_ instrucao	0	0	X	0	X	0	X	XX	0	X	0
decodificacao _Instrucao	0	0	X	**1**	X	0	X	XX	0	X	0
fim_desvio_ incondicional	0	**1**	X	**1**	X	0	X	XX	0	X	0
fim_instrucao	**1**	0	0	0	0	0	0	00	0	0	0

EXTENSOR DE SINAL

É uma unidade que recebi um sinal de 16 bits, e tem como saída 32 bits, sendo eles os 16 recebidos com um deslocamento de 16 bits a esquerda, e o outros 16 bits preenchidos com zero.

OPERAÇÃO ULA

É uma unidade que recebi dois sinais, sendo eles os últimos 6 bits da instrução e dois bits vidos da unidade de controle, o ULAOp.

STFR LEFT DOIS

ficamos de analisar a necessidade do mesmo

BANCO DE REGISTRADORES

É uma unidade composta por um conjunto de registradores que podem ser acessados de forma organizada. Podem ser executadas operações de leitura dos dados anteriormente gravados e de escrita de dados para modificar as informações internas. Este componente é um dos componentes mais importantes do fluxo

de dados em um processador. As informações que estão sendo processadas em um determinado momento devem estar armazenadas no banco de registradores.

ULA

É a unidade lógica e aritmética (ULA) ou em inglês Arithmetic Logic Unit (ALU) responsável por realizar operações lógicas e aritméticas.

MEMÓRIA DE DADOS

A unidade de memória é um elemento de estado com entradas para endereço e para o dado a ser escrito, e uma única saída para leitura de resultado.

AGORA OS SEIS MULTIPLEXADORES

MULTIPLEXADOR 32 BITS - Mux32_instrucao

Foi adicionado esse multiplexador para iniciar o processador com a primeira instrução, selecionando como entrada o “input1” vindo da própria Unidade de Controle (UC). A entrada do “input1” só é necessária para iniciar a primeira instrução do processador, as demais instruções virá do Registrador (registrador_proxima_instrucao), ou seja, o sinal do multiplexador deve está selecionado para o “input0”. Esse controle de seleção do sinal é feito pela UC.

MULTIPLEXADOR 5 BITS - Mux5_registrador_escrita

Nesse multiplexador é feito o controle do endereço de escrita do Banco de Registradores, esse endereço está contido no código de máquina da instrução que está sendo executada. Esse endereço pode vir dos bits [20..16] ou [15..11], a escolha de qual sequência de bits utilizar é feita pela Unidade de Controle com o sinal “RegDst”.

MULTIPLEXADOR 32 BITS - Mux32_desvio_condicional

Esse multiplexador é responsável por fazer o controle do endereço da próxima instrução a ser executada. A entrada “input1” vem do “somador_endereco” que recebe o endereço do PC+1 mais o valor do “extensor_de_sinal”, o resultado dessa soma é o endereço do desvio condicional. A entrada “input0” é o endereço vindo do PC+1. O controle de qual endereço será selecionado vem da combinação com uma porta AND dos sinais “DvC”, da Unidade de Controle, com o sinal “zero”, vindo da ULA.

MULTIPLEXADOR 32 BITS - Mux32_desvio_incondicional

Esse multiplexador faz o controle do endereço da próxima instrução a ser executada. Seu “input1” vem do Concatenador, que possui o endereço de desvio incondicional, e o “input0” vem do Multiplexador de Desvio Condicional (Mux32_desvio_condicional). A seleção é feita pelo sinal DVI vindo da Unidade de Controle.

MULTIPLEXADOR 32 BITS - Mux32_dataULA2

Esse multiplexador é responsável por selecionar o sinal de “data2” para a Unidade Lógica e Aritmética (ULA). A entrada “input0” vem do Banco de Registradores, o dado lido do registrador 2. O “input1” vem do “extensor_de_sinal”. A seleção é feita pelo sinal ULAFonte da Unidade de Controle.

MULTIPLEXADOR 32 BITS - Mux32_

MULTIPLEXADOR 32 BITS - Mux32_data_write

Esse multiplexador foi adicionado para resolver o problema da inserção de dados no Banco de Registradores. Como as instruções do Tipo R, e Tipo I precisam acessar dados que já estão armazenados no Banco de Registradores ou na Memória de Dados, era necessário forçar o dado de escrita por um sinal externo vindo do pino “data_write_registrador”. Contudo, a partir do momento que já existia dados escritos no Banco de Registradores ou na Memória de Dados, não era mais necessário utilizar o pino “data_write_registrador” pois as instruções do Tipo R e Tipo I já poderão retornar o valor de dado a ser escrito. A seleção de qual dado será escrito no Banco de Registradores é feita pelo pino “sel_data_write”