Relatório de Projecto para Sistemas Digitais Reconfiguráveis (SDR)

Tarefa 1 para área 4: construir um processador que vai permitir executar operações especificadas pelo professor.

André Fernandes Neto – n.m.:25201

Resumo: Recorrendo à linguagem VHDL pretende-se implementar numa FPGA Spartan3 um processador que permita realizar um dado número de operações. Implementa-se uma versão simples de um processador apenas o suficiente para suportar algumas operações do "instruction set" de um processador MIPS. Este é um processador com uma arquitectura "Von Neumann machine". Encontram-se implementadas as seguintes operações: add, addi, and, andi, beq, lw, or, ori, slt, slti, sub, sw.

Conteúdo

A modulação deste trabalho foi feita nos ficheiros apresentados na Tabela 2.

Ficheiro	Descripção
Alphabet.vhd	Contém a *package Alphabet*, onde se encontram definidos o desenho dos diversos caracteres visualizados no monitor. Esta definição é usada para iniciar a ROM contida no ficheiro SROM.vhd
ALU.vhd	Descrição da ALU usada pelo processador ao nível do comportamento (behavioural).
ALUctrlUnit.vhd	Implementação da unidade de controlo da ALU.
CRAM.vhd	Instanciação de uma block RAM e a sua inicialização. Esta é a memória "gráfica" onde é colocado o texto a aparecer no monitor.
ctrlUnit.vhd	Descrição da unidade de controlo do processador.
dataMemory.vhd	Implementação de uma memória para dados (operações: lw,sw). A sua descripção é feita ao nível comportamento (behavioural).
DETIUA-S3.ucf	Ficheiro UCF – define as ligações ao exterior
general.vhd	Este é o ficheiro de mais alto nível. Nele são instanciados todos os outros elementos/componentes do processador. É nele que se encontram também alguns elementos essenciais do processador (registos do processador).
InstructionMemory.vhd	Implementação de uma memória para as instruções (apenas de leitura e assíncrona). A sua descrição é feita ao nível comportamento (behavioural).

muxSIZE.vhd	Componente multiplexer 2x1, com barramento de dimensão configurável.
ProgCounter.vhd	Descrição do Program Couter do processador (síncrono).
regReader.vhd	Módulo usado para disponibilizar o valor do diversos registos para a memória CRAM.
signExtend.vhd	Componente que expande um sinal de 16 bits para 32bits, mantendo o seu sinal (propagando o bit mais significativo):
SROM.vhd	SROM memória usada para "imprimir" os diversos caracteres no monitor.
vga.vhd	Interface para monitor VGA.

Tabela 1 - Ficheiros criados e a sua descripção.

Estrutura do processador

A estrutura do processador segue o modelo descrito na Ilustração 1.

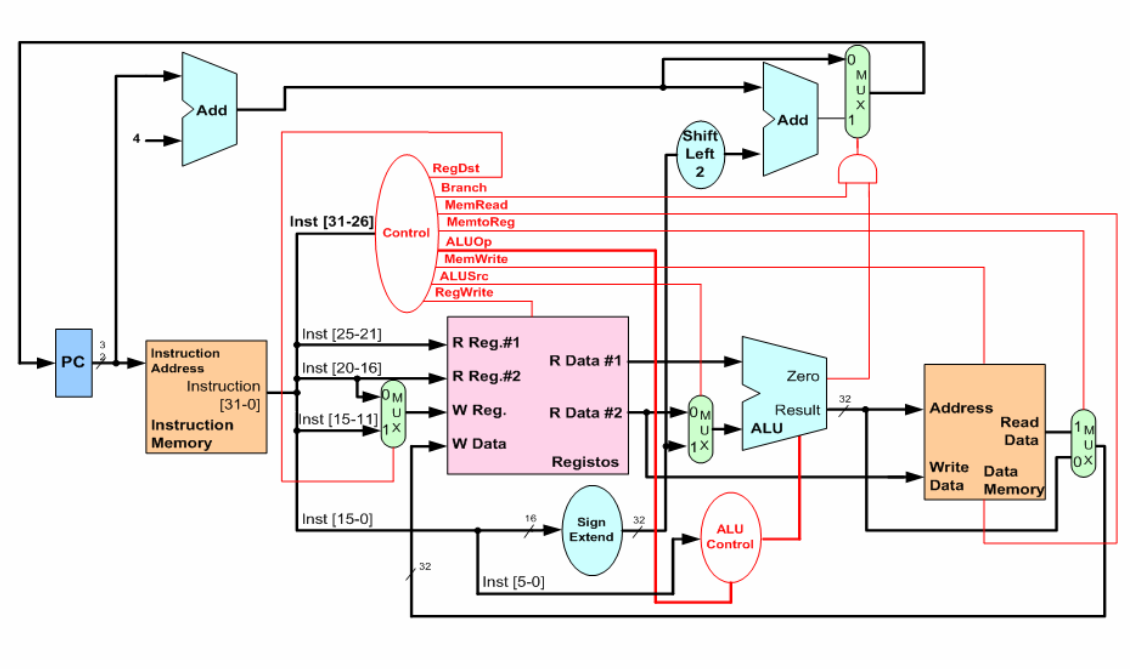

Ilustração 1 - Estrutura do processador

Componentes principais

ALU – descreve as principais características de uma ALU : add, sub, slt, and e or; possui um multiplexer descrito ao nível comportamental.

Registers – estão descriptos no módulo de mais alto nível, para facilitar o seu acesso por todos os outros componentes que deles necessitem. A escrita nos registos é feita de modo síncrono com o flanco ascendente do sinal de relógio, que corresponde ao final do ciclo da instrução.

PC – Program Counter, actualiza o seu valor no flanco ascendente do sinal de relógio.

Instruction Memory – Memória assíncrona, onde são colocadas as instruções a realizar pelo processador.

Control Unit – Unidade de controlo onde são gerados os sinais de controlo dos diversos componentes do processador. Excepto no caso da instruções tipo R em que os sinais de controlo da ALU são gerados na unidade ALU Control Unit.

ALU Control Unit – Unidade que gera os sinais de controlo da ALU com base no campo *function* das instruções tipo R, caso contrário usa os sinais de controlo já gerados pela Control Unit. Este processo não segue os métodos mais comuns de implementação, foi no entanto usado por uma questão de simplicidade.

Data Memory – Memória usada para armazenar e ler dados. Esta é uma memória síncrona, a leitura é feita no flanco descendente do sinal de relógio, que corresponde ao meio do ciclo da instrução, e a escrita é feita no flanco ascendente do sinal de relógio, que corresponde ao fim do ciclo da instrução.

Instruções suportadas:

ADD -- Add
Operação: $d = $s + $t;
Codificação: 0000 00ss ssst tttt dddd d000 0010 0000
ADDI – Add immediate
Operação: $t = $s + imm;
Codificação: 0010 00ss ssst tttt iiii iiii iiii iiii
AND -- Bitwise and
Operação: $d = $s & $t;
Codificação: 0000 00ss ssst tttt dddd d000 0010 0100
ANDI -- Bitwise and immediate
Operação: $t = $s & imm;
Codificação: 0011 00ss ssst tttt iiii iiii iiii iiii
BEQ -- Branch on equal
Operação: if $s == $t advance_pc (offset << 2)); else
Codificação: 0001 00ss ssst tttt iiii iiii iiii iiii
LW -- Load word
Operação: $t = MEM[$s + offset];
Codificação: 1000 11ss ssst tttt iiii iiii iiii iiii
NOOP -- no operation
Codificação: 0000 0000 0000 0000 0000 0000 0000 0000
OR -- Bitwise or
Operação: $d = $s | $t;
Codificação: 0000 00ss ssst tttt dddd d000 0010 0101
ORI -- Bitwise or immediate
Operação: $t = $s | imm;
Codificação: 0011 01ss ssst tttt iiii iiii iiii iiii
SLT -- Set on less than (signed)
Operação: if $s < $t $d = 1; advance_pc (4); else $d = 0;
Codificação: 0000 00ss ssst tttt dddd d000 0010 1010
SLTI -- Set on less than immediate (signed)
Operação: if $s < imm $t = 1; advance_pc (4); else $t = 0;
Codificação: 0010 10ss ssst tttt iiii iiii iiii iiii
SUB -- Subtract
Operação: $d = $s - $t;
Codificação: 0000 00ss ssst tttt dddd d000 0010 0010
SW -- Store word
Operação: MEM[$s + offset] = $t;
Codificação: 1010 11ss ssst tttt iiii iiii iiii iiii

Bibliografia:

- J.Hennessy, D.A.Patterson, *Computer Organization and Design – the hardware/software interface*, Elsevier, 2004
- Slides das aulas teóricas da disciplina de Arquitectura de Computadores I – DETI (Docente responsável: Nuno Lau)
- Slides disponibilizados no site da disciplina de SDR - http://www.ieeta.pt/~iouliia/Courses/SDR/index.html
- MIPS Instruction Reference http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html