UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO
MC542 – ORGANIZAÇÃO DE COMPUTADORES – TEORIA E PRÁTICA
2º Semestre de 2007
Prof. Paulo César Centoducatte

RELATÓRIO DO PROJETO PRÁTICO

Haraldo Sérgio Albergaria Pereira Filho (003059)
Jônatas Botelho da Silva (033546)

1 – OBJETIVO

O objetivo deste trabalho foi projetar a versão simplificada (sem mecanismo de detecção de hazard e forwarding, porém reduzindo a penalidade devido ao hazard de controle a um ciclo) do processador MIPS pipeline capaz de executar no mínimo as instruções *lw (load)*, *sw (store)*, *add*, *addu*, *addi*, *sub*, *subu*, *subi*, *and*, *or*, *xor*, *slt*, *sltu*, *j*, *jal* e *beq* utilizando a linguagem VHDL e a ferramenta GHDL. Foi projetado também um *testbench* que foi utilizado para simular o processador utilizando a ferramenta GTKWAVE.

2 – DESCRIÇÃO

2.1 - Entidade

O processador MIPS pipeline foi projetado com base na entidade abaixo:

```
Entity mips is
generic(nbits : positive := 32);
  port(Instruction     : in  std_logic_vector(nbits -1 downto 0);
       Data            : in  std_logic_vector(nbits -1 downto 0);
       clk             : in  std_logic;
       reset           : in  std_logic;
       PCF             : out std_logic_vector(nbits -1 downto 0);
       ALUOutM         : out std_logic_vector(nbits -1 downto 0);
       WriteDataM      : out std_logic_vector(nbits -1 downto 0);
       MemWriteM       : out std_logic);
End mips;
```

2.2 – Componentes

A arquitetura estrutural da entidade *mips* foi descrita no arquivo *mips.vhd* e instancia os seguintes componentes, cujas entidades estão descritas no arquivo *pkg_mips.vhd*:

Unidade de Controle:

A descrição comportamental da Unidade de Controle foi feita no arquivo *control.vhd*, com base na unidade de controle projetada no exercício 3, sendo que a maior modificação é que não há o alu_decoder, sendo que ALUCntr <= Funct(3 downto 0). A Unidade de Controle implementa a seguinte tabela verdade:

Instrução	Opcode (5:0)	RegDst	RegWrite	MemtoReg	MemWrite	Branch	Jump	AluSrc	AluControl (3:0)
R-type	000000	1	1	0	0	0	0	0	Funct(3:0)
Load	100011	0	1	1	0	0	0	1	0000
Store	101011	X	0	X	1	0	0	1	0000
Addi	001000	0	1	0	0	0	0	1	0000
Subi	001001	0	1	0	0	0	0	1	0010
Beq	000100	X	0	X	0	1	0	0	0010
J	000010	X	0	X	0	X	1	X	XXXX
Jal	000011	X	1	X	0	X	1	X	XXXX

ULA:

A descrição comportamental da ULA foi feita no arquivo *alu.vhd*, com base na ULA projetada no exercício 1. Neste projeto, a ULA foi instanciada 3 vezes: na ULA principal, no cálculo do PC+4 e no cálculo do PC Branch, sendo que nos dois útlimos casos ela funciona como um somador sinalizado (Funct(3:0) = “0000”). A ULA implementa a seguinte tabela verdade:

Funct(3:0)	Função	Tipo
0000	A + B (signed)	Aritmética
0001	A + B (unsigned)	Aritmética
0010	A – B (signed)	Aritmética
0011	A – B (unsigned)	Aritmética
0100	A and B	Lógica
0101	A or B	Lógica
0110	A xor B	Lógica
0111	A nor B	Lógica
1010	SLT (signed)	Slt
1011	SLT (unsigned)	Slt

Banco de Registradores:

A descrição comportamental do banco de registradores foi feita no arquivo *rf.vhd*, sendo que foi utilzado o mesmo do exercício 2.

Extensor de sinal:

A descrição comportamental do extensor de sinal foi feita no arquivo *signext.vhd*, sendo responsável por fazer a extensão do sinal do imediato das instruções I-type.

Multiplexador de 2 entradas:

A descrição comportamental do multiplexador de 2 entradas foi feita no arquivo *mux2.vhd*, sendo implementado um multiplexador com entradas parametrizadas com valor padrão de 32 bits. Foi instanciado 7 vezes, sendo 5 com o valor padrão de 32 bits e 2 com parâmetro de 5 bits.

Registrador:

A descrição comportamental do registrador foi feita no arquivo *reg.vhd*, sendo implementado um registrador com entradas parametrizadas com valor padrão de 32 bits. Foi instanciado 5 vezes, implementando o pipeline, com 32, 64, 115, 72 e 71bits, fazendo a transição entre os estágios PCF/PC, F/D, D/E, E/M e M/W respectivamente.

2.3 – Conjunto de Instruções

Foram implementadas as instruções *lw (load), sw (store), add, addu, addi, sub, subu, subi, and, or, xor, nor, slt, sltu, j, jal* e *beq* de acordo com os campos Opcode e Funct das tabelas verdade da unidade de controle e da ULA respectivamente, sendo que os mesmos foram retirados do Apêndice B do livro "Digital Design and Computer Architeture". A única exceção é a instrução *subi*, que não constava do apêndice e que utilizou o opcode da instrução *addiu*, que não foi implementada.

2.4 – Tratamento de Hazard

Não foi implementado tratamento de hazard, ou seja, o processador não faz forwarding e/ou stall. Apesar disso, a penalidade devido ao hazard de controle nas instruções *beq*, *j* e *jal* foi reduzido a um ciclo, fazendo o a verificação do "branch taken" e do cálculo do novo PC no estágio de decodificação.