MIPS VHDL Overview

Reference: VHDL Tutorial on CDROM, or Accolade Reference Guide
http://www.acc-eda.com/vhdref

Notes: / 3055-05 / pdf / MIPS_vhdl_notes.pdf

Single-Cycle MIPS Model (Patterson & Hennessy fig. 5.39)

MIPS (after changes to make it pipelining)
Top Level - MIPS.vhl - includes "control" Ports and Connections to other modules

ARCHITECTURE structure OF MIPS IS

COMPONENT control
PORT:
  Opcode : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
  RegDst : OUT STD_LOGIC;
  ALUSrc : OUT STD_LOGIC;
  MemtoReg : OUT STD_LOGIC;
  MemWrite : OUT STD_LOGIC;
  MemRead : OUT STD_LOGIC;
  Branch : OUT STD_LOGIC;
  ALUOp : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
  clock, reset : IN STD_LOGIC;
END COMPONENT;

...  

COMPONENT control
PORT MAP:
  Opcode => ALUSrc;
  RegDst => ALUSrc;
  ALUSrc => ALUSrc;
  MemtoReg => MemtoReg;
  MemWrite => MemWrite;
  MemRead => MemRead;
  Branch => Branch;
  ALUOp => ALUOp;
  clock => clock;
  reset => reset;
END control;

BLUE - Output Port of "control," RED - Input Port of "control," GREEN - Internal Signals.

MIPS modules for Lab-3

END control ;
PORT MAP (
-- control module (implements MIPS control unit)
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY control IS
PORT(
  ENTRY Opcode: IN STD_LOGIC_VECTOR( 5 DOWNTO 0 );
  ENTRY RegDst: OUT STD_LOGIC;
  ENTRY ALUSrc: OUT STD_LOGIC;
  ENTRY MemtoReg: OUT STD_LOGIC;
  ENTRY RegWrite: OUT STD_LOGIC;
  ENTRY MemRead: OUT STD_LOGIC;
  ENTRY MemWrite: OUT STD_LOGIC;
  ENTRY Branch: OUT STD_LOGIC;
  ENTRY ALUop: OUT STD_LOGIC_VECTOR( 1 DOWNTO 0 );
  ENTRY clock, reset: IN STD_LOGIC);

END control;

These parameters must match the list in MIPS.vhd in order and type. Matching names as well avoids confusion.

MIPS "control" module - (2 of 2)

ARCHITECTURE behavior OF control IS

SIGNAL R_format, Lw, Sw, Beq: STD_LOGIC;

BEGIN
  R_format <= '1' WHEN Opcode = "000000" ELSE '0';
  Lw <= '1' WHEN Opcode = "100011" ELSE '0';
  Sw <= '1' WHEN Opcode = "101011" ELSE '0';
  Beq <= '1' WHEN Opcode = "000100" ELSE '0';
  RegDst <= R_format;
  ALUSrc <= Lw OR Sw;
  MemtoReg <= Lw;
  RegWrite <= R_format OR Lw;
  MemWrite <= Sw;
  Branch <= Beq;
  ALUOp(1) <= R_format;
  ALUOp(0) <= Beq;

END behavior;

BLUE - Output Port of "control," RED - Input Port of "control," GREEN - Internal

"idecode" module - external Ports (1 of 4)

-- idecode module (implements the register file for
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY idecode IS
PORT(
  ENTRY read_data_1: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
  ENTRY read_data_2: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
  ENTRY Instruction: IN STD_LOGIC_VECTOR( 31 DOWNTO 0 );
  ENTRY RegWrite: IN STD_LOGIC;
  ENTRY MemWrite: IN STD_LOGIC;
  ENTRY Sign_extend: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
  ENTRY clock, reset: IN STD_LOGIC);

END idecode;
ARCHITECTURE behavior OF idecode IS

TYPE register_file IS ARRAY ( 0 TO 31 ) OF STD_LOGIC_VECTOR( 31 DOWNTO 0 );

-- "register_file" is an array of 32-bit logic vectors (e.g., 32-bit words)

SIGNAL register_array : register_file;

SIGNAL write_register_address : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
SIGNAL write_data : STD_LOGIC_VECTOR( 31 DOWNTO 0 );
SIGNAL read_register_1_address : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
SIGNAL read_register_2_address : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
SIGNAL write_register_address_1 : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
SIGNAL write_register_address_0 : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
SIGNAL Instruction_immediate_value : STD_LOGIC_VECTOR( 15 DOWNTO 0 );

BEGIN

read_register_1_address <= Instruction ( 25 DOWNTO 21 );
read_register_2_address <= Instruction ( 20 DOWNTO 16 );
write_register_address_1 <= Instruction ( 15 DOWNTO 11 );  -- for Reg-Reg operations
write_register_address_0 <= Instruction ( 20 DOWNTO 16 );  -- for other operations (no read_2)
Instruction_immediate_value <= Instruction ( 15 DOWNTO 0 );

read_data_1 <= register_array ( CONV_INTEGER ( read_register_1_address ) );  -- Read Reg1
read_data_2 <= register_array ( CONV_INTEGER ( read_register_2_address ) );  -- Read Reg2

write_register_address <= write_register_address_1;  -- Mux for Register Write Address
WHEN RegDs = '1' ELSE write_register_address_0;  -- weird syntax for: if(...) ... else ...

-- Mux to bypass data memory for Rformat instructions
write_data <= ALU_result ( 31 DOWNTO 0 )
WHEN ( MemtoReg = '0' ) ELSE read_data;

Sign_extend <= X"0000" & Instruction_immediate_value  -- Sign Extend 16-bits to 32-bits
WHEN Instruction_immediate_value(15) = '0'  -- depends on left-most bit
ELSE X"0000" & Instruction_immediate_value;

PROCESS  -- this is "clocked logic", last page was
BEGIN

WAIT UNTIL clock'EVENT AND clock = '1';

END IF;  -- no change in register_array() unless one of two conditions satisfied
END PROCESS;

END behavior;
MIPS modules for Lab-3 and labeling delayed signals

* Change clock phase for "Data Memory" and Req-Set "Write_Data" (write on negative edge)