REGISTER FILE DESIGN AND SIMULATION

CLASS PROJECT SUMMARY

Instruction set architecture describes the programmer's view of the machine. This level of computer blueprinting allows design engineers to discover expected features of the circuit including:

- data processing instructions like ADD, SUB, AND, EOR, etc.,
- memory load-store instructions like LDR, STR, etc.,
- branch instructions with both conditional and unconditional flow control,
- the names and sizes of the registers provided for computational storage and flow control,
- the size of data memory provided for longer term storage during program execution, and
- the binary encodings for each instruction.

These features are then used by design engineers as they choose components to organize together into a working circuit. Multiple **micro-architectures** are possible for any given instruction set architecture because different design engineers can choose different components and different organizational strategies when implementing the features. In the end, however, any micro-architecture design must implement the features described by the instruction set architecture.

One organizational decision that leads to different micro-architectures is the number of clock periods used per instruction. The three common clock-period strategies are called ***single-cycle***, ***multi-cycle***, and ***pipelined***.

- Single-cycle processors use one clock-period per instruction and the clock-period is set by the total delay of the slowest instruction. This is a disadvantage as faster instructions cannot execute more quickly. The advantage, however, is straightforward control circuitry.
- Multi-cycle processors use multiple clock-periods per instruction and each instruction uses the minimum number of clock periods required for its execution. This allows faster instructions that do not access data memory, like ADD, to avoid the unnecessary delay of the data memory stage. Thus, the advantage is speed for faster instructions. The disadvantage is more complex control because a finite state machine controller must be built to coordinate control signals across multiple clock periods.
- Pipelined processors exploit instruction level parallelism to allow multiple instructions to be in execution at the same time. This is accomplished by adding state registers between the instruction fetch, instruction decode, execute, memory access, and write-back stages of the circuit.

The CE1921 laboratory is designed as a large multi-week project requiring students to design and simulate a ***single-cycle processor*** for a subset of the ARMv4 instructions. Students are required to:

- **Design** VHDL and schematic data path components including registers, a register file, instruction ROM, data memory, ALU, extenders, and controllers.
- **Organize** the components together into a top-level schematic that implements a single-cycle processor.
- **Simulate** the processor using a basic test program.

REGISTER FILE DESIGN AND SIMULATION

LABORATORY EXERCISE

The ARMv4 instruction set architecture uses a CPU register file with sixteen 32-bit registers named R0 through R15. Instructions encode register addresses as 4-bit fields within the instruction binary number using names such as Rd, Rn and Rm. These address bitfields are connected to **three address inputs of the register file**. Address A1 selects the register to output as RD1 and become input A of the ALU. Address A2 selects the register to output as RD2 and potentially become input B of the ALU. Address A3 selects which register should store the data calculated by the circuit if the controller commands a store using the REGWR control signal. Figure 1 shows the register file in position within the arithmetic circuit.

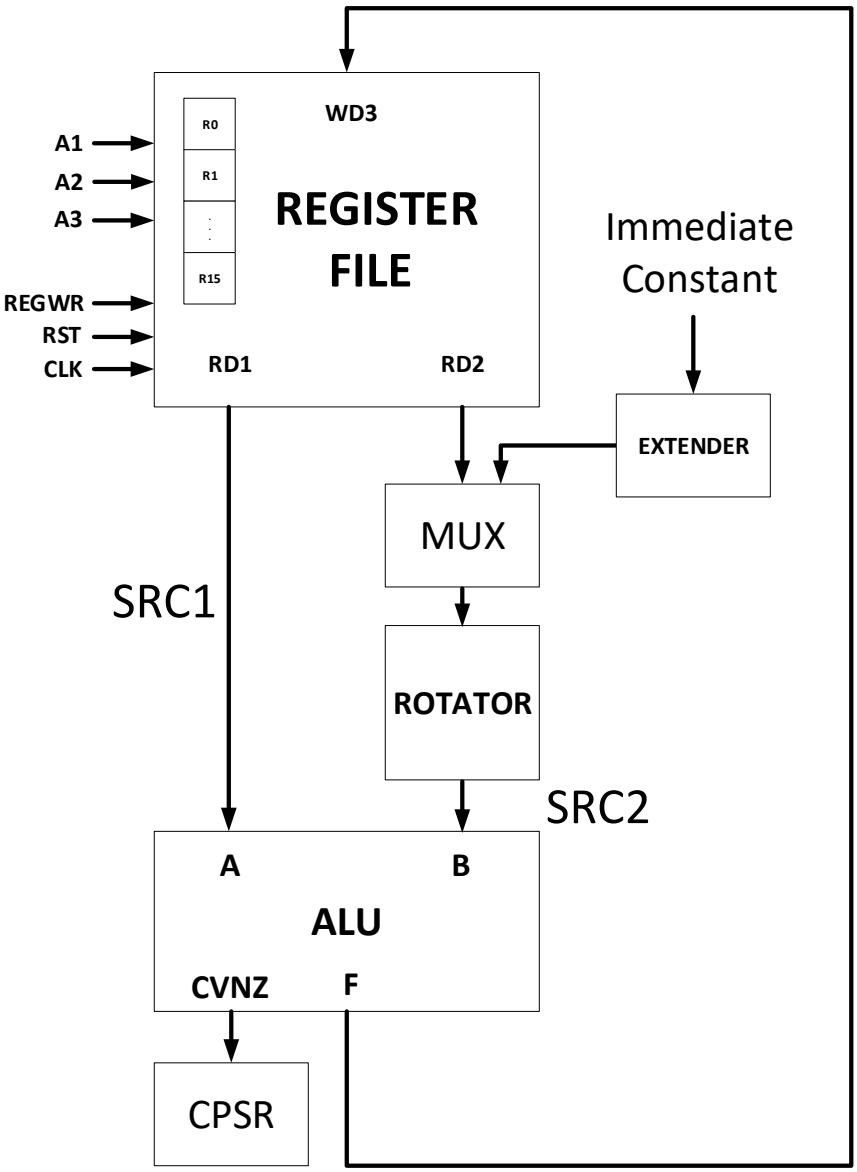
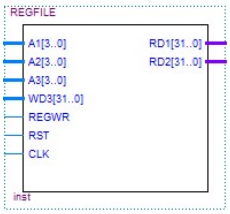


Figure 1: The CE1921 ARMV4 ARITHMETIC CIRCUIT

REGISTER FILE DESIGN AND SIMULATION

Use the skeleton code provided on the next page to guide your work as you **complete** the VHDL description for the CE1921 register file component.

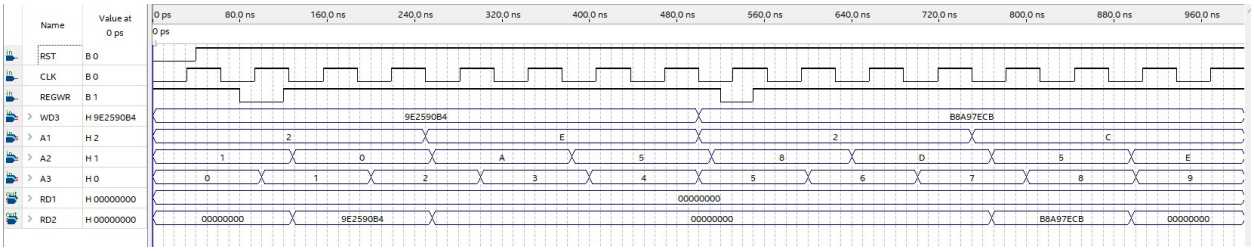
COMPONENT	BEHAVIOR
	RD1 <= register requested by address A1 RD2 <= register requested by address A2 register requested by address A3 <= WD3 if active-low REGWR is active all registers synchronously reset when active-low RST is active

SIMULATION REQUIREMENTS

- Overwrite a 62.5ns clock through simulation time.
- Reset through the first rising edge.
- Write two random arbitrary binary value onto WD3 at 500 ns intervals.
- Write four random binary value on A1 through simulation time.
- Write eight random binary values on A2 through simulation time.
- Overwrite a count sequence on A3 at 100ns intervals.
- Choose a couple of places to insert REGWR commands.
- Run the simulation.
- Explain your simulation results.
- Repeat three more times for a total of four captured simulation runs.

SUBMISSION

- You must submit well-commented VHDL code and simulation waveform diagrams using your instructor's preferred submission method. Simulation can be completed using a Quartus university waveform file or a Quartus VHDL testbench with Modelsim-Altera waveform results.
- You must comment on how you know the simulation is correct.
- This example shows random values illustrating how the simulation should look.



"I know that this simulation is correct because at around 100 nanoseconds the A2 input requests register R0 be output as RD2. This outputs the WD3 data that was earlier written to register R0 at about 85 ns…"

REGISTER FILE DESIGN AND SIMULATION

```
-- ****************************************************************
-- * project:     regfile
-- * filename:    regfile.vhd
-- * author:      << insert your name here >>
-- * date:        MSOE Spring Quarter 2020
-- * provides:    a register file for the CE1921 processor
-- ****************************************************************

-- use library packages
--  std_logic_1164: 9-valued logic signal voltages
--  std_logic_numeric_std: allows arithmetic on std_logic_vectors
library ieee;
use ieee.std_logic_1164.all;

-- function block symbol
-- inputs:
--    A1, A2: 4-bit addresses specifying output registers RD1 and RD2
--    A3    : 4-bit address specifying register to write WD3 data to
--    WD3   : 32-bit data to be stored in register addressed by A3
--    REGWR : control signal to determine if input WD3 data gets stored
--    RST   : active-low synchronous reset signal
--    CLK   : clock for synchronized register behavior
-- outputs:
--    RD1   : 32-bit output from register specified by address A1
--    RD2   : 32-bit output from register specified by address A2
entity REGFILE is
port(A1    : in std_logic_vector(3 downto 0);
     A2    : in std_logic_vector(3 downto 0);
     << FINISH DECLARING THE SIGNALS IN THE ORDER SHOWN ABOVE >>

end entity REGFILE;

-- circuit description
architecture BEHAVIORAL of REGFILE is
   -- declare internal signals that will become register outputs
   signal R0,R1,R2,R3,R4,R5,R6,R7,R8: std_logic_vector(31 downto 0);
   << declare the next 16 registers R9 through R15 as 32-bit wide registers >>

begin

  -- use A1 and A2 to control with-select multiplexers for
  -- outputs RD1 and RD2
  with A1 select
  RD1 <= R0 when B"0000",
         R1 when B"0001",
         R2 when B"0010",
         << complete this multiplexer >>

         R15 when others;

  with A2 select
  << complete >>  <= R0 when B"0000",
         R1 when B"0001",
         << complete this multiplexer >>
         R15 when others;

  -- implement sixteen registers with active-low synchronous reset
  -- and active synchronous load when A3 specifies the register
  reg0: process(rst,clk)
  begin
    if rising_edge(clk) then
      if RST = '0' then R0 <= X"00000000";
      elsif REGWR = '0' then
        if A3 = B"0000" then R0 <= WD3;
```

```
        end if;
      end if;
    end if;
  end process;

  reg1: process(rst,clk)
  begin
    if rising_edge(clk) then
      if RST = '0' then R1 <= X"00000000";
      elsif REGWR = '0' then
        if A3 = B"0001" then R1 <= WD3;
        end if;
      end if;
    end if;
  end process;

  << complete sixteen registers >>

end architecture BEHAVIORAL;
```