ALU 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:

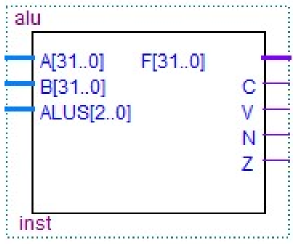
- **Design** VHDL and schematic data path components including registers, a register file, instruction ROM, data memory, ALU, extenders, and controllers.
- **Organize** the components together into a top-level schematic that implements a single-cycle processor.
- **Simulate** the processor using a basic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

ALU DESIGN AND SIMULATION

LABORATORY EXERCISE

The heart of any instruction set processor is its arithmetic circuit. The width of the arithmetic circuit sets the **word size** of the machine. This word size is used to set the size of all other architectural features including the width of registers, the data bus, the address bus, and instruction binary numbers. The ARMv4 instruction set architecture specifies a word size of 32 bits.

Use the skeleton code provided on the next page to guide your work as you **complete** the VHDL description for the CE1921 ARMv4 ALU. The skeleton code uses a package created by Synopsys, Incorporated that extends arithmetic to standard logic vectors.

COMPONENT	BEHAVIOR
 alu A[31..0] B[31..0] ALUS[2..0] F[31..0] C V N Z inst	ALUS F ------------------------ 0 A+B 1 A-B 2 A AND B 3 A OR B 4 A XOR B 5 A 6 B 7 1 ALU Arithmetic Condition Flags Carry, Signed number line oVerflow, Negative, and Zero C = carry out of the computation = F(32) using the approach below. V = see circuit equation truth table in the skeleton code N = sign of the result = F(31) Z = 1 if F = 0

SIMULATION REQUIREMENTS

- Write one arbitrary value onto A and B through simulation time.
- Write a count sequence on S through simulation time.
- Simulation verifies that F demonstrates correct calculation, pass, or constant 1 values.
- Simulation verifies that C,V,N, and Z are correct for the arbitrary values on A.
- See the final page of this laboratory for an example.

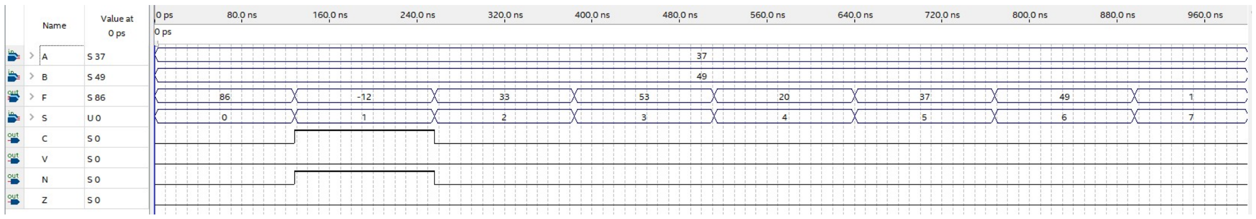
ALU DESIGN AND SIMULATION

SKELETON CODE

```
-- *****************************************************************
-- * project:     alu
-- * filename:    alu.vhd
-- * author:      << insert your name here >>
-- * date:        MSOE Spring Quarter 2020
-- * provides:    32-bit ALU for the CE1921 single-cycle 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;
use ieee.std_logic_unsigned.all;


-- function block symbol
-- A, B, and F are 32-bit numbers
-- signed and unsigned only matter when doing comparison
-- the ALU doesn't do comparison so declaring as unsigned is fine
entity ALU is
port(A,B: in std_logic_vector(31 downto 0);
     S: in std_logic_vector(2 downto 0);
     F: out std_logic_vector(31 downto 0);
     C,V,N,Z: out std_logic);
end entity ALU;

-- circuit description
architecture MULTIPLEXER of ALU is

  -- 32-bit addition results in carry-out to column 32.
  -- use 33-bit vectors to store that extra carry-out bit
  signal INTA, INTB, INTF: std_logic_vector(32 downto 0);

begin

  -- connect INTA and INTB to the inputs
  INTA(32) <= '0';
  INTA(31 downto 0) <= A;
  INTB(32) <= '0';
  INTB(31 downto 0) <= B;

  -- complete the arithmetic and logic
  with S select
  INTF <= INTA+INTB when B"000", -- addition
          INTA-INTB when B"001", -- subtraction
                                -- COMPLETE the rest of the with-select

  -- connect the 32-bit result to the output signal
  F <= INTF(31 downto 0);

  -- create the flag bits to announce particular result events
  C <= INTF(32); -- the carry out to column 32

  N <=           -- COMPLETE: negative is just the MSB of the 32-bit result

  Z <= '1' when INTF(31 downto 0) = X"00000000" else '0';

  -- COMPLETE the truth table for signed overflow
  --   overflow occurs when two positives add to a negative
  --   overflow occurs when two negatives add to a positive
  --   overflow occurs when negative minus positive gives positive
  --   overflow occurs when positive minus negative gives negative
  --   remember that the sign bit of each number is is bit 31
  --
  -- COMPLETE TRUTH TABLE BY FILLING IN VALUES FOR V
  -- USING SELECT SIGNAL S(0) TO DIFFERENTIATE BETWEEN ADD AND SUBTRACT
  --
  -- S(0) INTA(31) INTB(31) INTF(31) | V   COMPLETE COMMENTS BEHIND V ALSO
  -- ------------------------------------
  -- 0    0        0        0        | 0   + plus + = +
  -- 0    0        0        1        | 1   + plus + = -
  -- 0    0        1        0        | 0   + plus - = +
  -- 0    0        1        1        |
  -- 0    1        0        0        |
  -- 0    1        0        1        |
  -- 0    1        1        0        |
  -- 0    1        1        1        |
  --
  -- 1    0        0        0        | 0   + minus + = +
  -- 1    0        0        1        | 0   + minus + = -
  -- 1    0        1        0        | 0   + minus - = +
  -- 1    0        1        1        |
  -- 1    1        0        0        |
  -- 1    1        0        1        |
  -- 1    1        1        0        |
  -- 1    1        1        1        |


  -- IMPLEMENT V AS A CANONICAL LOGIC EQUATION
  V <= (not S(0) and not INTA(31) and not INTB(31) and INTF(31)) or -- minterm 1
                                                                    -- COMPLETE minterms

end architecture MULTIPLEXER;
```

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. Be sure to simulate multiple times and include all diagrams – including one that causes a Z-flag condition to be true.
- You must comment on how you know the simulation is correct.
- You are not allowed to use the same arbitrary numbers as this example.



"I know that this simulation is correct because all values reflect the expected calculations selected using S. For example, 37+49 = 86 and 37 is passed as the output when S=6. I've provided a table below that shows the work that verifies each of the selection results …."

"I have also checked the flag outputs. Here is how I know that they are correct…"