INSTRUCTION ROM 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.

INSTRUCTION ROM DESIGN AND SIMULATION

LABORATORY EXERCISE

The ARMv4 fetch circuit is responsible for reading the current instruction from the instruction memory and for calculating the control flow numbers needed to advance the program counter to the next instruction. Instruction memories are often flash ROM memories in ARM microcontrollers used in embedded systems. Figure 1 shows the CE1921 ARMv4 fetch circuit with a 32-bit address provided to the instruction ROM by the program counter register. It also shows two adders used to calculate new program counter values for use by other circuit stages. The lower adder creates PC+4 because ARMv4 instructions are 32-bit wide numbers and thus the next instruction is four bytes away in memory. The upper adder creates PC+8 for use in branch address calculations.

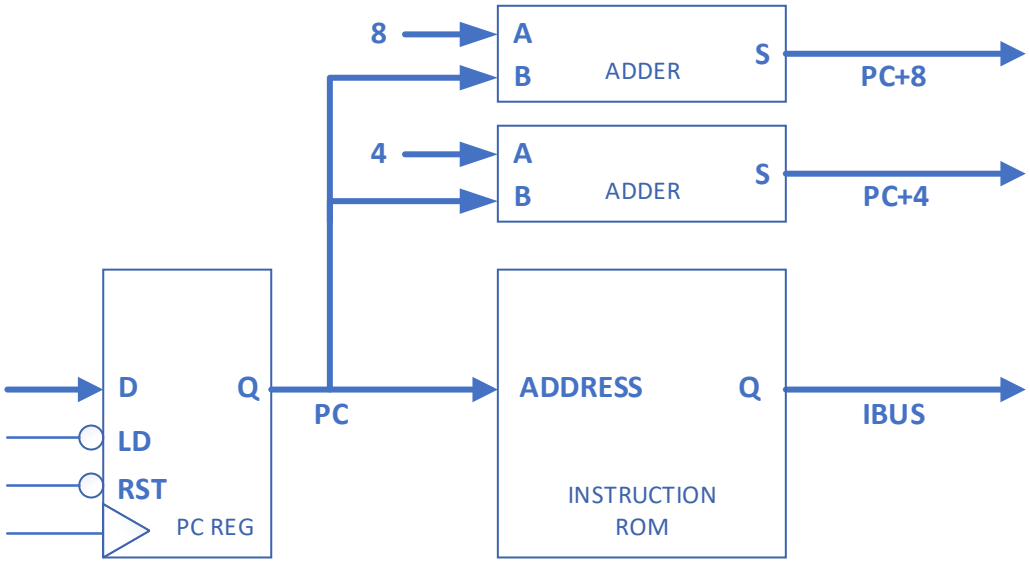


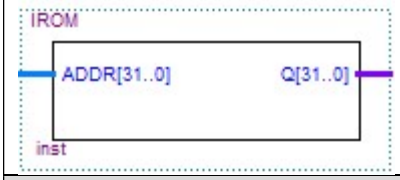
Figure 1: The CE1921 ARMV4 INSTRUCTION FETCH CIRCUIT

The first program that will run on the CE1921 ARMv4 processor is described by this instruction ROM memory image.

IROM ADDRESS	MACHINE CODE INSTRUCTION
0000 0000	E3A0 800A
0000 0004	E3A0 9000
0000 0008	E358 0000
0000 000C	0A00 000B
0000 0010	E089 9008
0000 0014	E248 8001
0000 0018	E358 0000
0000 001C	1AFF FFF9
0000 0020	E3A0 A000
0000 0024	E24A A020
0000 0028	E009 A00A
0000 002C	E35A 0000
0000 0030	0A00 0002
0000 0034	E3A0 B001
0000 0038	E3A0 C004
0000 003C	E58C B000
0000 0040	E59C 6000
0000 0044	EAFF FFFD

INSTRUCTION ROM 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 instruction ROM component.

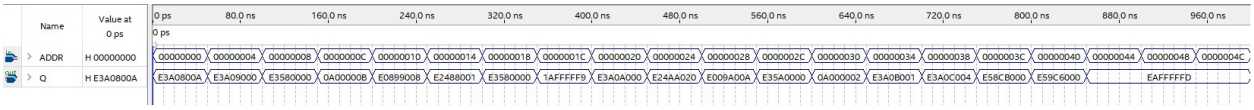
COMPONENT	BEHAVIOR
	Q <= machine code instruction specified by the address input

SIMULATION REQUIREMENTS

- Insert the ADDR and Q groups in hexadecimal.
- Overwrite a count sequence that increments by 4 at 50ns intervals.
- Run the simulation.
- Explain your simulation results.

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 how the simulation should look.



"I know that this simulation is correct because I checked every value against the machine code table. For example, at address … the machine code instruction should be …. and you can see in the simulation that it is. Similarly, …" "I completed Table 1 in this report that shows I validated every instruction machine code is correct."

INSTRUCTION ROM DESIGN AND SIMULATION

```
-- ********************************************************************
-- * project:     irom
-- * filename:    irom.vhd
-- * author:      << insert your name here >>
-- * date:        MSOE Spring Quarter 2020
-- * provides:    an instruction ROM for the CE1921 processor
-- ********************************************************************

-- use library packages
--  std_logic_1164: 9-valued logic signal voltages
library ieee;
use ieee.std_logic_1164.all;


-- function block symbol
-- inputs:
--    ADDR  : 32-bit address requesting instruction
-- outputs:
--    Q     : 32-bit output of machine code instruction
-- notes    : ROMs do not reset on power-up so no reset signal
--          : ROMs do not load in user mode so no load signal
entity IROM is
port(ADDR  : in std_logic_vector(31 downto 0);
     Q     : out std_logic_vector(31 downto 0));
end entity IROM;

-- circuit description
architecture MULTIPLEXER of IROM is
begin

  -- use address to output correct binary machine code number
  with ADDR select
  Q <= X"E3A0_800A" when X"0000_0000",    -- mov r8,#10
       X"E3A0_9000" when X"0000_0004",    -- mov r9,#0
       << complete remaining machine code numbers >> -- also hand disassemble and include
                                                     -- the assembly instruction as shown
       X"EAFF_FFFD" when others;          -- b 0x00000040

end architecture MULTIPLEXER;
```