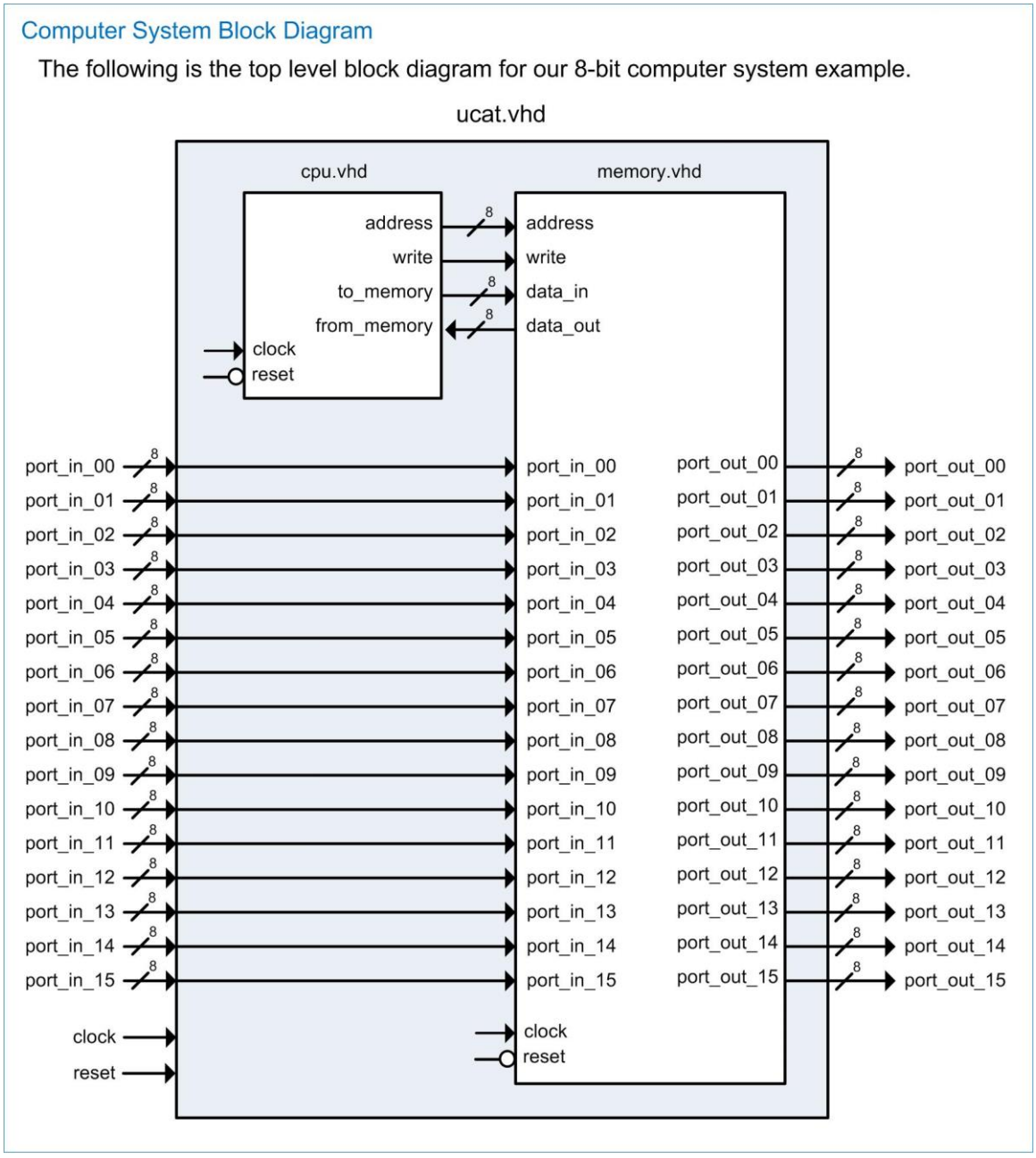
EELE 367 – Logic Design
Project Description

"The µCat Computer System"

Project Description

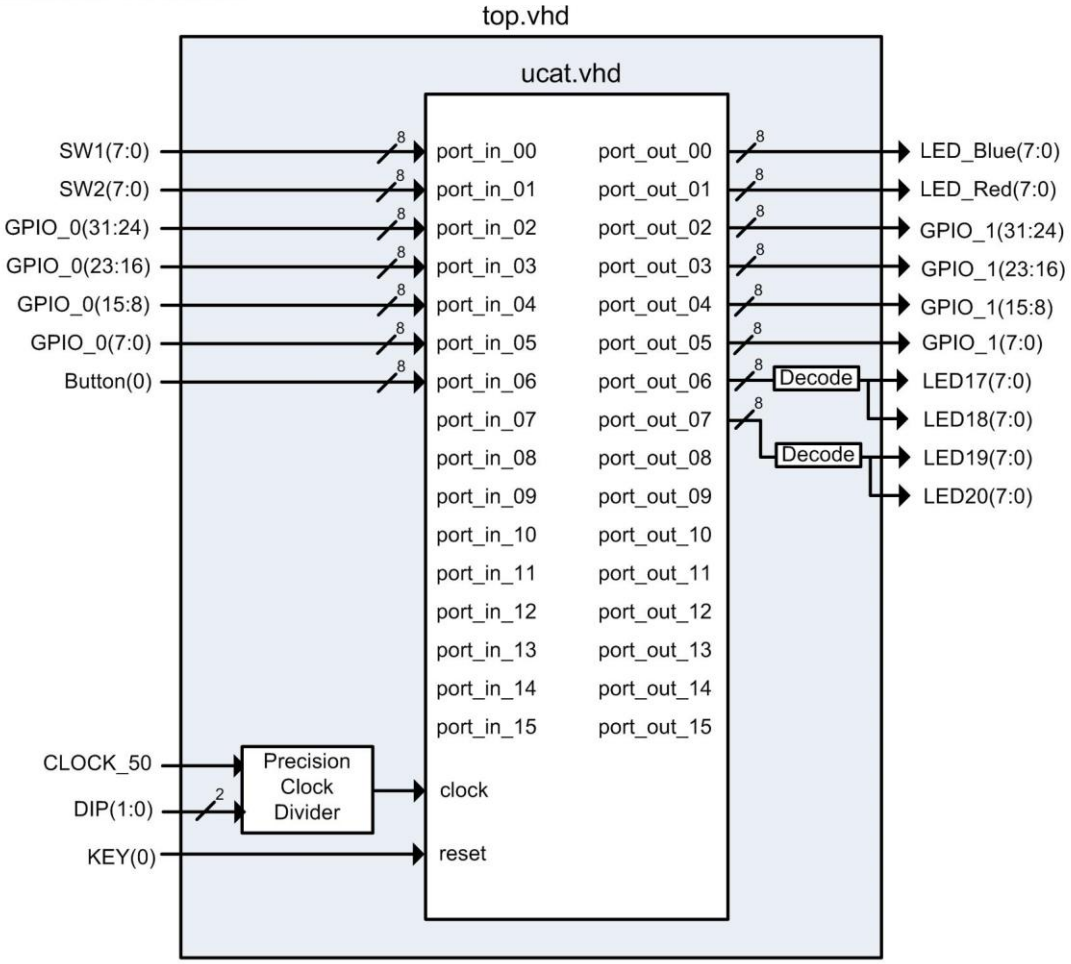
Instead of a final exam, you are going to design, implement and document a full computer system. The computer system will be provided as a fully functional product to a user that can program it in using assembly language. The computer system, called "The µCat" (that's micro cat…), will have an 8-bit processor, 128 bytes of program memory, 96 bytes of data memory (e.g., RAM), 16x 8-bit output ports, and 16x 8-bit inputs ports. The I/O ports of the computer are mapped to a variety of peripherals on the DE0-nano and I/O shield. The following figure shows the block diagram of the uCat computer system.

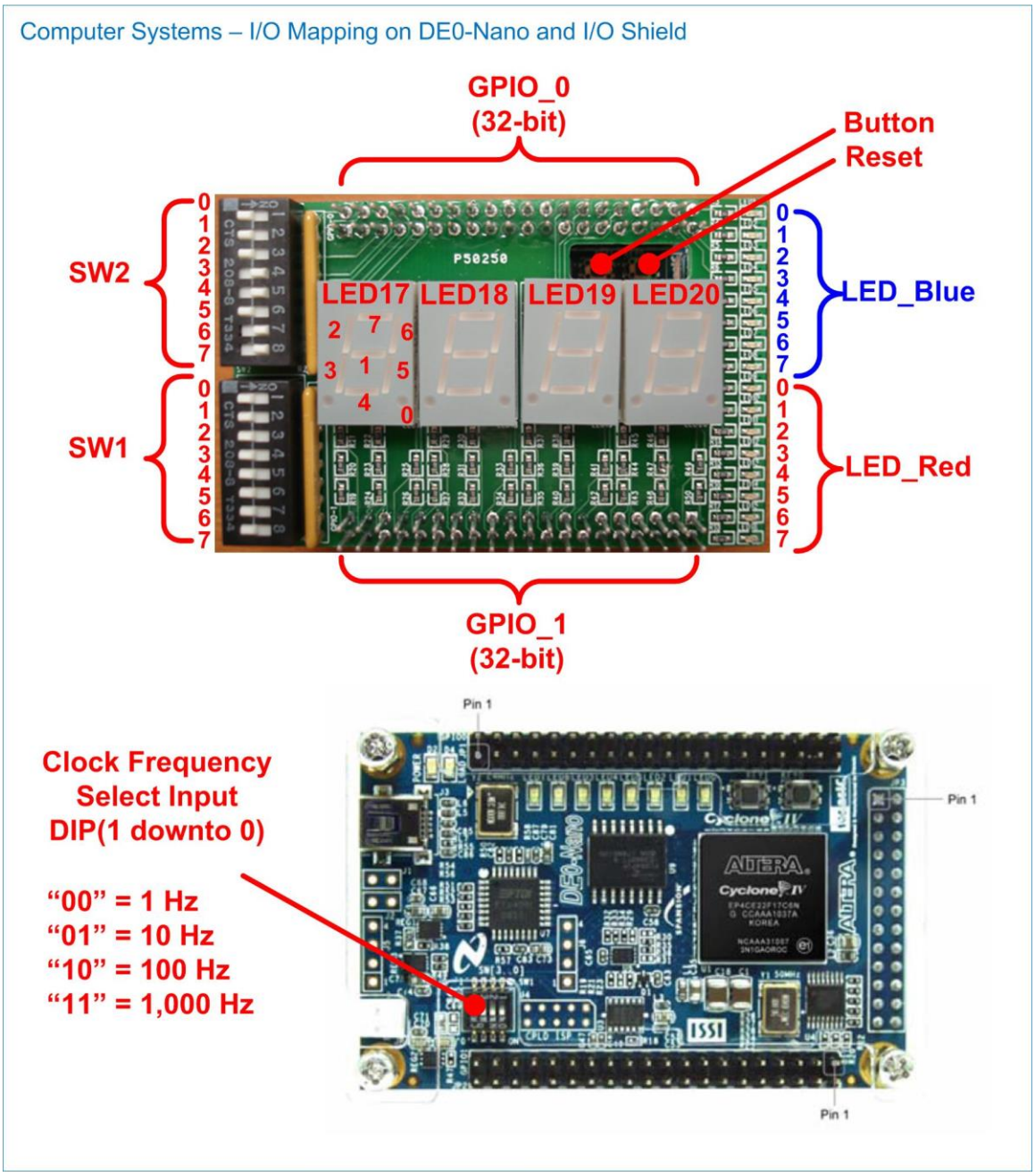


Once verified using ModelSim simulations, the uCat will be implemented on the DE0-nano board where its ports will be connected to the I/O as follows:

DE0-Nano I/O Mapping

The following shows how the I/O of the uCat computer interfaces to the I/O on the DE0-nano and I/O Shield.

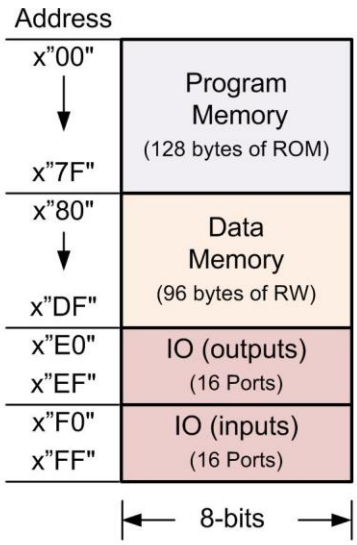




The memory system for the uCat is comprised of read only *Program Memory* (128 bytes), read/write *Data Memory* (96 bytes), output ports (16x), and input ports (16x). The memory system is designed such that everything is available to the CPU as an address. The following shows the memory map and the block diagram of the memory system.

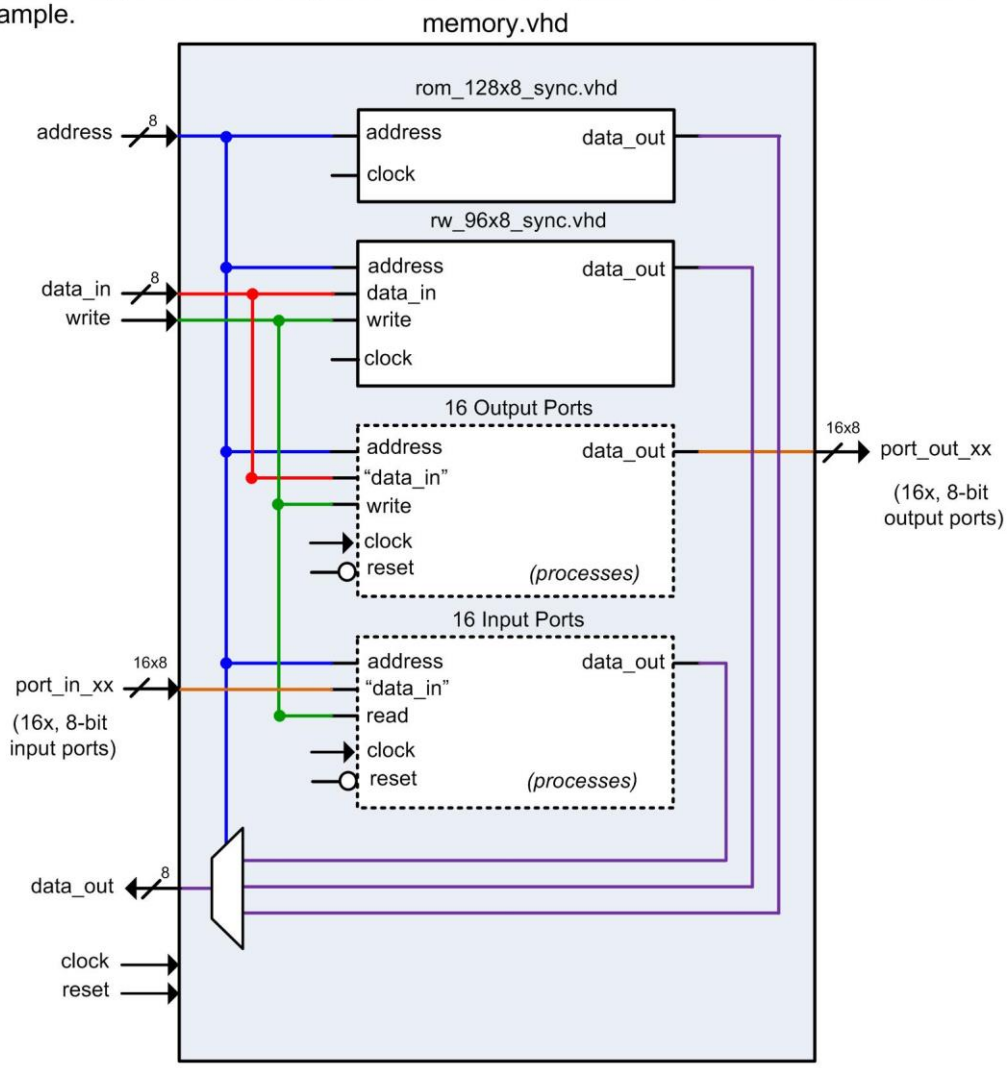
Memory Map

The following is the memory map for our 8-bit computer system example.



Memory System Block Diagram

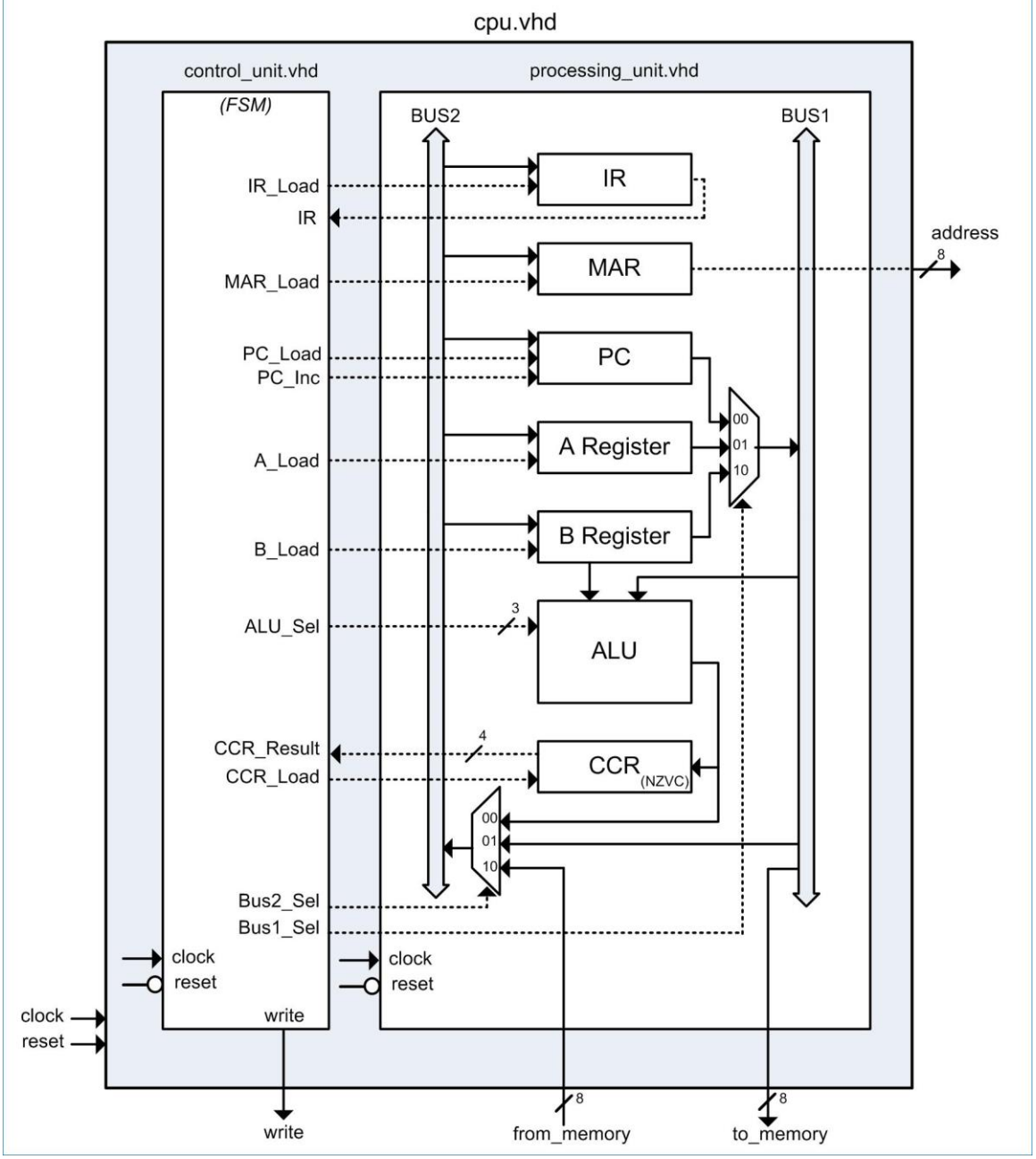
The following is the block diagram for the memory system of our 8-bit computer system example.



The CPU contains two 8-bit registers available to the programmer (A and B), an Arithmetic Logical Unit (ALU) to perform math and logic operations, and a sequence controller state machine that handles the details of performing an instruction (e.g., Fetch, Decode, Execute). The following is the block diagram of the CPU.

Central Processing (CPU) Block Diagram

The following is the block diagram for the CPU of our 8-bit computer system example.



The CPU will be designed to execute the following instructions.

Instruction Set

The following is a base set of instructions that the 8-bit computer system will be able to perform. Each instruction is given a descriptive pneumonic that will allow the system implementation and programming to be more intuitive. Each instruction is also provided with a unique Opcode. Some instructions have an Operand, which provides additional information necessary for the instruction. If an instruction contains an Operand, a description is provided as to how it is used (e.g., as data or as an address).

Pneumonic	Opcode, Operand	Description
LDA_IMM	x"86", *<data>*	Load Register A using Immediate Addressing
LDA_DIR	x"87", *<addr>*	Load Register A using Direct Addressing
STA_DIR	x"96", *<addr>*	Store Register A to Memory using Direct Addressing
LDB_IMM	x"88", *<data>*	Load Register B with Immediate Addressing
LDB_DIR	x"89", *<addr>*	Load Register B with Direct Addressing
STB_DIR	x"97", *<addr>*	Store Register B to Memory using Direct Addressing
BRA	x"20", *<addr>*	Branch Always to Address Provided
BEQ	x"21", *<addr>*	Branch to Address Provided if Result is equal to Zero
ADD	x"42"	A = A + B (plus)
SUB	x"43"	A = A - B (minus)
AND	x"44"	A = A B (AND)
OR	x"45"	A = A + B (OR)
INCA	x"46"	A = A + 1 (plus)
INCB	x"47"	B = B + 1 (plus)
DECA	x"48"	A = A - 1 (minus)
DECB	x"49"	B = B – 1 (minus)

The functionality of the registers within the CPU are as follows:

Instruction Register (IR) - This will hold the Opcode of the current instruction being executed.

Memory Address Register (MAR) – This will hold the current address being used to access memory. This can come from either the program counter when retrieving an instruction, or from the operand of an instruction in the case of direct addressing.

Program Counter (PC) – This will hold the address of the next location in program memory to retrieve an Opcode or Operand.

A & B – Two, 8-bit registers under the control of the programmer.

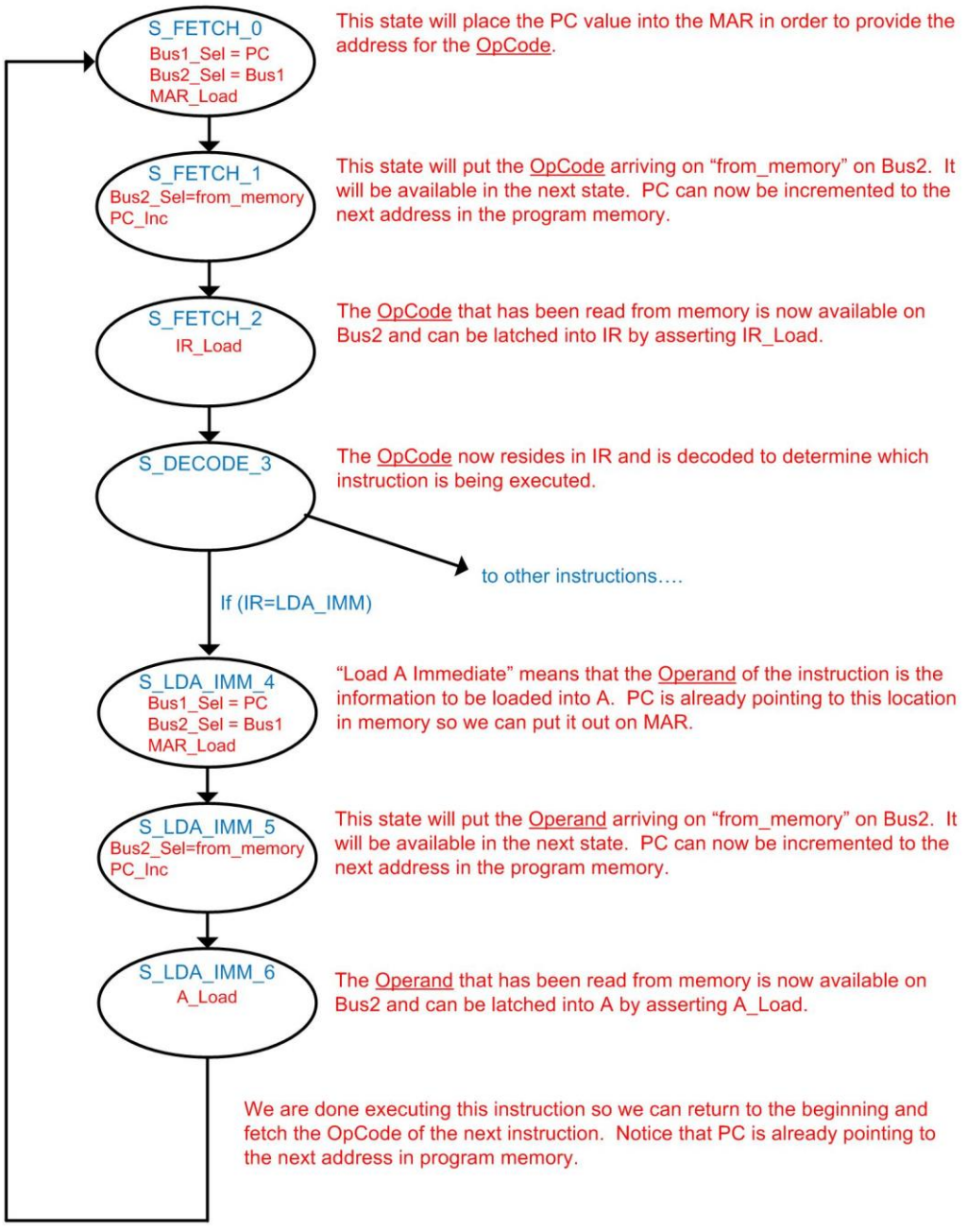
Arithmetic Logical Unit ALU) – A combinational logic circuit that performs addition, subtraction, logical AND, logical OR, increments and decrements. The inputs to the ALU come from Bus1 and register B. The output of the ALU goes to the multiplexer driving Bus2. The functionality of the ALU is dictated by the ALU_Sel lines (3 bits) where “000”=Add, “001”=Subtract”, “010”=AND, “011”=OR, “100”=Increment”, and “101”=Decrement.

Condition Code Register (CCR) – This register holds status information about the results of an ALU operation. The CCR_Result output contains a negative flag (N), a zero flag (Z), a two’s complement overflow flag (V), and a carry flag (C).

The following are the state diagrams that describe how the CPU executes three essential instructions (LDA_IMM, STA_DIR, and BRA).

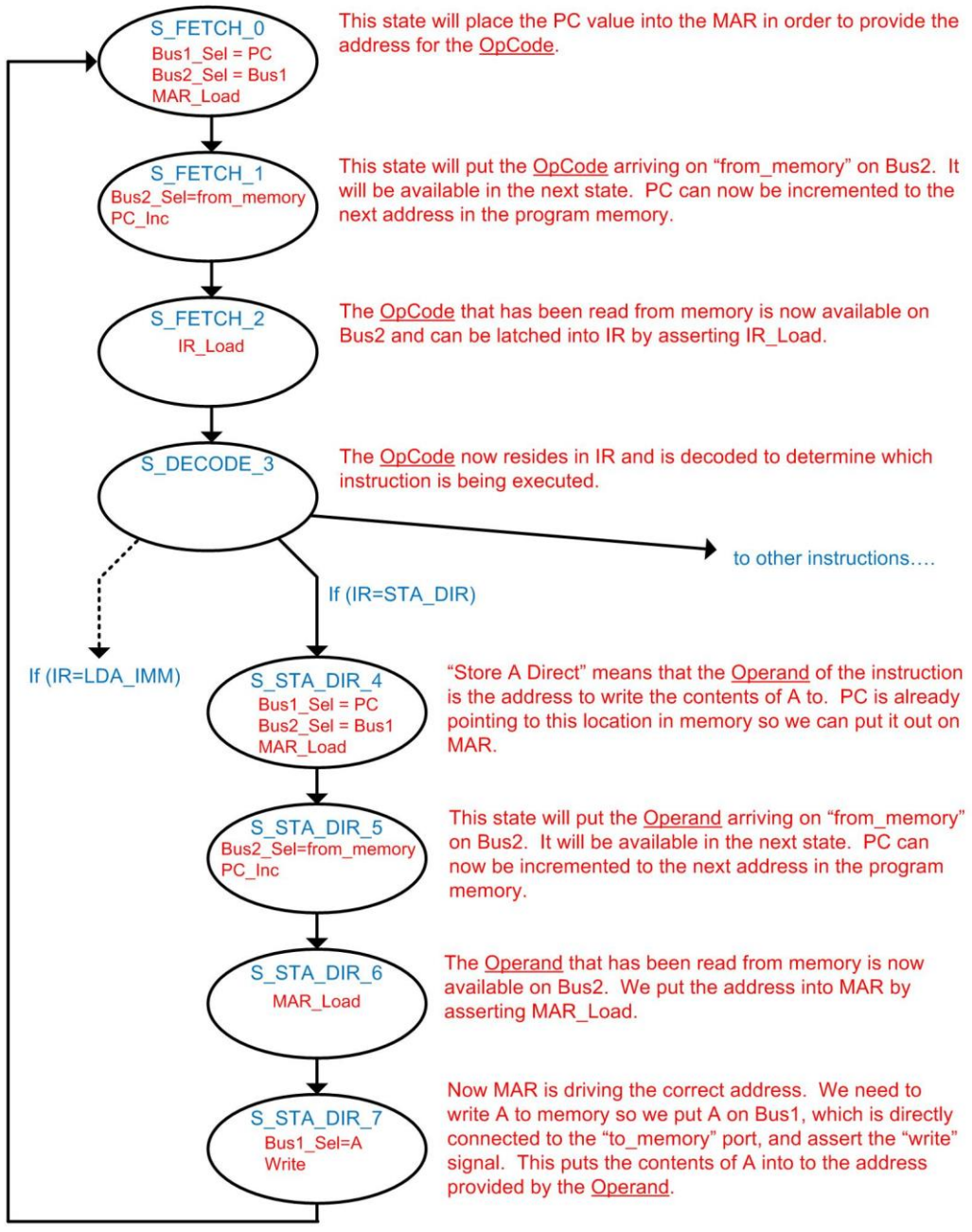
State Diagram for Instructions "Loading A with Immediate Addressing" (LDA_IMM)

The following is the state diagram for LDA_IMM. This load instruction will move information from memory into register A. Immediate addressing implies that the information to be put into A is provided as the operand of the instruction.



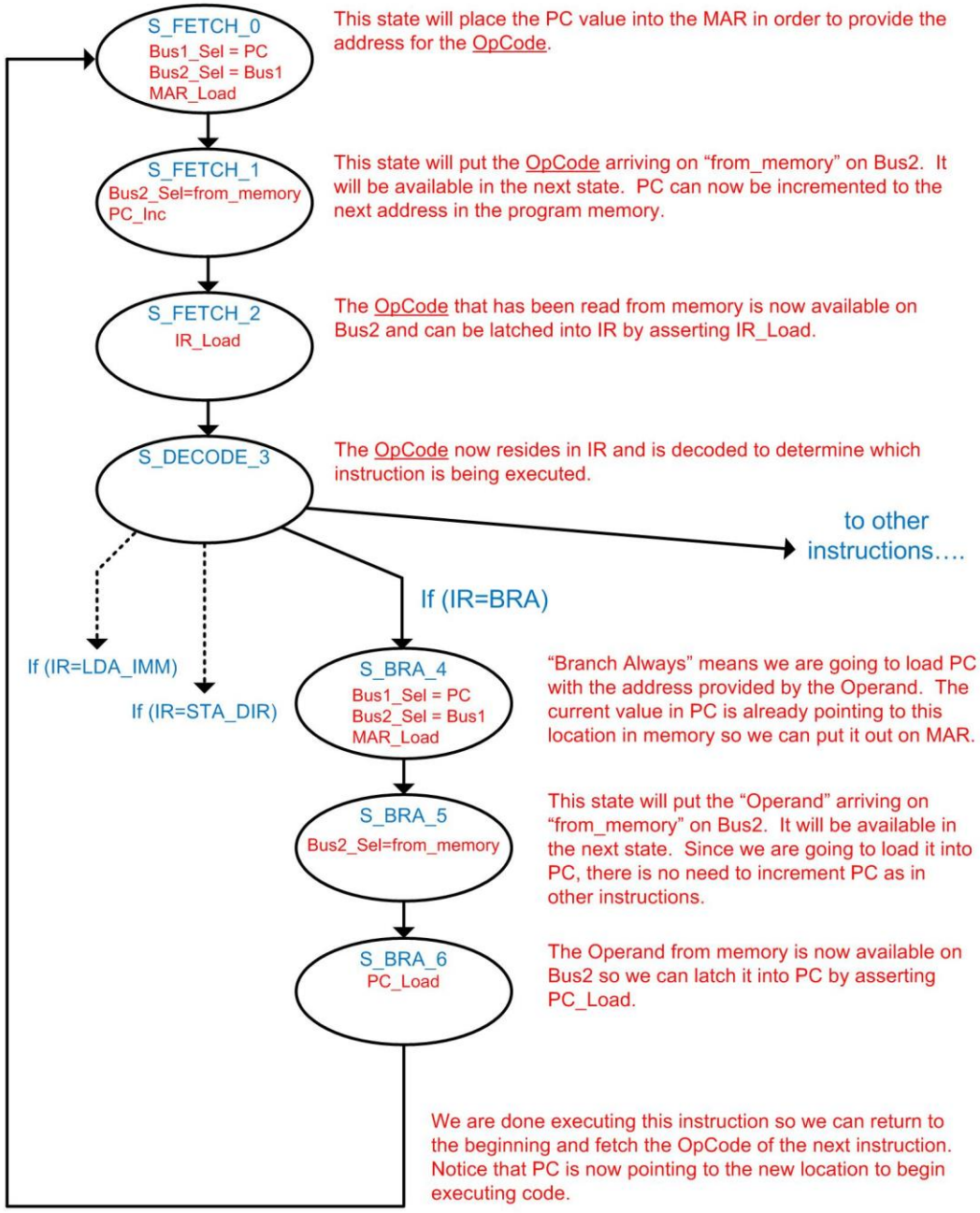
State Diagram for Instructions "Storing A with Direct Addressing" (STA_DIR)

The following is the state diagram for STA_DIR. This store instruction will move information from register A to memory. Direct addressing implies that the address that will receive the information from A is provided as the operand of the instruction.



State Diagram for Instruction "Branch Always" (BRA)

The following is the state diagram for BRA. This instruction will load the program counter (PC) with the address supplied by the operand of the instruction. This has the effect of setting the location of the next instruction to be executed.



Grading & Deliverables

This project has a series of deliverables, each with an increasing level of points associated with them. You may choose to complete as many, or as few as you'd like in order to achieve a desired score.

Part 1– 60%

For this part, you will implement the three instructions detailed in the provided state diagrams (LDA_IMM, STA_DIR, and BRA). You will implement the necessary VHDL for the uCat for these instructions and test your design with a simple program that loads A and then stores it to a variety of addresses (RAM and Ports). You will simulate your design with ModelSim to verify its functionality. This step is performed on just the ucat.vhd system. Then you will implement your design on the DE0-Nano board using the I/O mappings provided in the top.vhd system. You will demonstration the computer system writing various constant values to the character displays and LEDs of the I/O shield. Your deliverables for this part are:

- ModelSim Waveforms (annotated) showing the proper execution of each instruction (40%)
- A lab demonstration on the DE0-nano system showing various patterns being written to the character displays and LEDs of the DE0-nano (20%).

Part 2– 20%

Now implement all of the remaining instructions in the set EXCEPT any instructions using the ALU (e.g,. Add, Sub, AND, OR, Inc, Dec). You will perform ModelSim simulations verifying that each instruction is operational. You will then implement the computer system on the DE0-nano and show that it can read from the switches and write to the character displays and LEDs (verifying load direct and store direct are functional). Your deliverables for this part are:

- ModelSim Waveforms (annotated) (10%)
- A lab demonstration on the DE0-nano (10%)

Part 3– 20%

Now implement the ALU functions. You will verify proper operation using ModelSim simulations and then implement your system on the DE0-nano. Your lab demonstration for this part will consist of two programs. The first will implement a simple binary counter displayed on the character displays and LEDs. The second program will add SW1 to SW2 and display the sum on the LEDs and character displays. Your deliverables for this part are:

- ModelSim Waveforms (annotated) (10%)
- A lab demonstration on the DE0-nano (10%)

Extra Credit

After you have completed the above parts, you can keep going! Just discuss with me other ideas that you'd like to implement (e.g., more instructions, an advanced program, a stack, etc… The sky is the limit.