EE 231 - Homework 7

Due Oct. 24, 2016

Do the Prelab below for *Lab 8: Computer Control Unit (CCU)*

Abstract

At the core of all computers is a control unit. It provides mechanisms the procedurally stepping through instructions. When executing an instruction the computer must step trough multiple states. The task of stepping the computer through states and generating the necessary signals at each state is the purpose of the control unit. With some care, the system can be configured to work through a list of instructions, rather than just one. This forms the basis of software processors.

Prelab

Lab is nearing a first computer, but the control unit must first be built. A conceptual block diagram of a simple computer is shown in Figure 1. In previous labs the DATA MUX, the ALU, and required registers were already built. The control unit is a finite state machine. Its inputs are the instruction register and the carry as well as a clock pulse and `Reset`. The control units outputs are the control signals that direct the operation of the rest of the computer. The control unit can be in one of four states: `RESET`, `FETCH`, `EX1` and `EX2`.

- `RESET` is the reset state. The computer gets into this state when the `Reset` input is low and stays in this state until the `Reset` input goes high.
- `FETCH` is the fetch cycle. The computer program is stored in memory. During the fetch cycle the next instruction is fetched from memory and loaded into the instruction register (`IRX`).
- `EX1` is the first execution cycle. Once an instruction has been loaded into `IRX`, the control unit determines the required course of action to take based on the value of `IRX` and the current state of the control unit.
- `EX2` is the second execution cycle. Some instructions only require one execution cycle (`EX1`) while others require two (`EX1`, and `EX2`).

1. The output of the control unit depends on both the present state and the input. **What type of state machine is this?**
2. Draw the state diagram for the control unit.

3. Assign op codes to each instruction in the instruction set (Table 1.) **Justify your design choices**; thought now, can simplify problems later.

   - To improve readability, use **`parameter`(s)** to assign values that are frequently used in your program, e.g., op codes.
   - You should also provide default values for the control signals.

4. Write a Verilog program to implement the control unit.

Figure 1: Processor Block Diagram

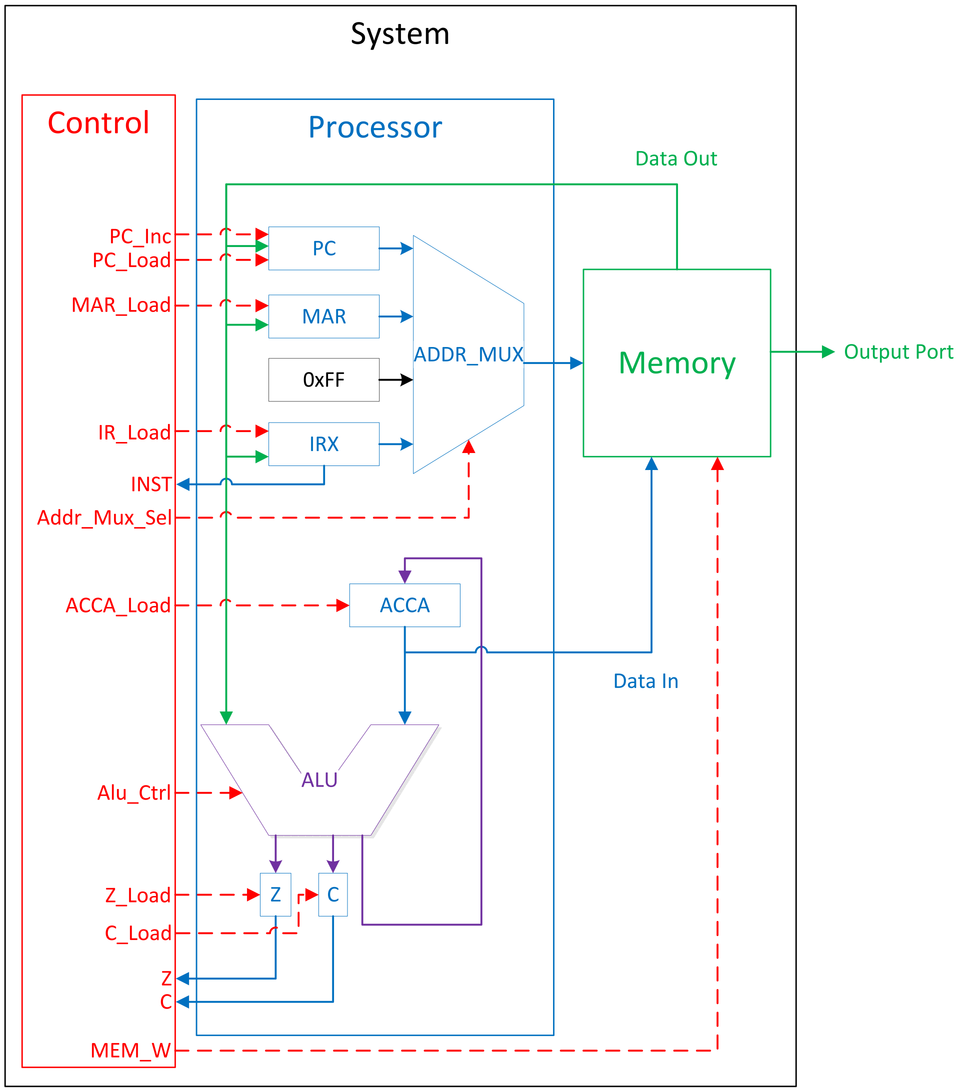

Table 1: Computer Instructions

Op.Code	**Instruction**	**Operation (Mnemonic)**
	`nop`	Do nothing. (No Operation)
	`LDDA addr`	Loads `ACCA` with the value in memory at address `addr`. `C` stays the same, `Z` changes. (Load `ACCA` from memory)
	`LDDA_IMM #num`	Loads `ACCA` with `num`, the value in memory at the address immediately following the `LDAA #num` command. `C` stays the same, `Z` changes. (Load `ACCA` with an immediate)
	`STAA addr`	Stores the value in `ACCA` at memory address `addr`. `C` stays the same, Z changes. (Store `ACCA` in memory)
	`ADDA addr`	Adds the value in memory location `addr` to the value in `ACCA` and saves the result in `ACCA`. `C` and `Z` change. (Add `ACCA` and value in memory)
	`SUBA addr`	Subtracts the value in memory location `addr` from the value in `ACCA` and saves the result in `ACCA`. `C` and `Z` change. (Subtract value in memory from `ACCA`)
	`ANDA addr`	Perform a logical `AND` of the value in memory location `addr` with the value in `ACCA`. Save the result in `ACCA`. `C` stays the same, `Z` changes. (Logical `AND` of `ACCA` and value in memory)
	`ORAA addr`	Perform a logical `OR` of the value in memory location `addr` with the value in `ACCA`. Save the result in `ACCA`. `C` stays the same, `Z` changes. (Logical `OR` of `ACCA` and value in memory)
	`CMPA addr`	Compare `ACCA` to value in `addr`. This is done by subtracting the value in `addr` from `ACCA`. `ACCA` does not change. `C` and `Z` change. (Compares `ACCA` to the value in `addr`)
	`COMA`	Replace the value in `ACCA` with its one's complement. `C` is set to 1 and `Z` changes. (Compliment `ACCA`)
	`INCA`	Increment value in `ACCA`. `C` stays the same and `Z` changes. (`INCA ACCA`)
	`LSLA`	Logical shift left of `ACCA`. `C` and `Z` change. (Logical shift left `ACCA`)
	`LSRA`	Logical shift right of `ACCA`. `C` and `Z` change. (Logical shift right `ACCA`)
	`ASRA`	Arithmetic shift right of `ACCA`. `C` and `Z` change. (Arithmetic shift right `ACCA`)
	`JMP addr`	Jumps to the instruction stored in address `addr`. The `PC` is replaced with `addr`. `C` and `Z` stay the same. (Jump)
	`JCS addr`	Jumps to the instruction stored in address `addr` if $C = 1$. If `C` is not set, continue with next instruction. `C` and `Z` stay the same. (Jump if carry set)
	`JCC addr`	Jumps to the instruction stored in address `addr` if $C = 0$. If `C` is set, continue with next instruction. `C` and `Z` stay the same. (Jump if carry not set)
	`JEQ addr`	Jumps to the instruction stored in address `addr` if $Z = 1$. If `Z` is not set, continue with next instruction. `C` and `Z` stay the same. (Jump if `Z` set)