The correct functionality of the CPU lies in perfect timing. The decoder must make sure every register and module receive the correct data in the appropriate cycle. Data is constantly fed in through the inputs. In addition to the current instruction, the current state, and the contents of the status register, a single bit stack overflow input was also added. Indicating if the stack register has reached its maximum value. This is helpful in the event of a stack overflow. At this point, the decoder will turn off the clock and halt the program, just as it would for an STP instruction.

In order to make the Verilog code easier to read and write, we have made wires for each instruction and assigned them to their corresponding opcodes. Different addressing modes have their bitfields allocated in separate ways. For example, the conditional field starts at various bits depending on the instruction’s mode. Therefore, it proved to be handy to have separate wires for each addressing type. Thus, we were able to easily create a 4-bit wire always containing the right information. Within an always block a case statement assigned then the correct fields of the status register for each possibility. The cond\_evaluated wire was AND-ed with certain control lines. This prevents the CPU from overwriting any register or data in the RAM if the condition specified in the current instruction is not being met.

Multiplexers are used in the top-level design to select the correct data as an input to modules such as the RAM, ALU or the registers. Their control lines are implemented in the decoder as well. It is not sufficient to list the instruction when writing the Boolean expressions, each element must be AND-ed with the correct state as well. For example, in the case of POP, in EXEC1 Rs is updated. However, that same input is later used to load the data from the RAM to the destination register in EXEC2. For larger multiplexers with 2 select lines, we assigned control buses. The main input to the register file is the one labelled as "data1". Data to this port can come from several places. For arithmetic instructions, this is the ALU, for MOV, it comes from output1 of the register file. Information can also come from the RAM. In the special case of LDA, a forcing block is required since the opcode must be discarded. Memory accessing commands will load data directly from the outputs of the RAM. The other input port of the register file is only used by multiplication and move word operations. For these instructions, only one destination register is specified. The incremented address comes from a separate port of the ALU. Write\_address and write\_enable select lines for both ports are assigned to choose the correct register in each addressing mode. Due to pipelining the program counter is always loaded with an incremented address. Thus, it will not require a multiplexer. Although this is not necessary, we strove to set default values for the control lines in states where their value did not matter.

The ALU needs to perform different arithmetic operations based on the instruction. To reduce the number of control lines and for simplicity, we have created a 6-bit encoded opcode. This will be passed on to the ALU. Each bit was assigned with the correct combination of the instruction wires.