This document discusses revisions that can be made to my 8MP16 and includes a more thorough description of the ALU organization than was presented in the original report.

**Relative Addressing**

In order to prevent the calculation of a relative address from overwriting the contents of the accumulator, the component labelled PCadd can be revised to be an adder that accepts the PC as the first input and either the constant 2 or the X field of the instruction as the second input. The only additional contral signal would be the one for the multiplexor that selects between 2 (for incrementing the PC) or X (for displacing the PC as a result of a successful branch instruction). In the concrete RTN for branch instructions, A ← B+X would be changed to PC2 ← B+X and PC ← A to PC ← PC2.

**Waiting for Input**

Just as a Wait signal is used for memory access, a Wait signal should also be present during the input stage of the INPUT instruction to allow time for the value to become available. The only change required is that the Wait signal be activated in the control word for state 7. It was not necessary during demonstration due to the use of break-points, which allowed for the input value to be set at any time during the instruction.

**Demonstration without Break-Points**

The on-board clock of the Altera DE2 can be used for demonstration if the Done switch is used as the catalyst for instruction execution. Upon setting the Start switch, the machine would pass through the first stage automatically and arrive at the memory access stage where it awaits the Done signal from the switch. When Done is activated, it proceeds to the remaining stages and then back to fetch (unless a Stop prevented it from doing so in the previous instruction). The cycle repeats, directed by the setting of the Done switch, once per instruction, with the exception of the INPUT instruction, which requires a second Done signal in the final stage when it waits for an input value to be available. This use of the Done signal is only possible because Done resets itself after the state in which it is set. Otherwise, if its 'set' value relied only on the position of the on-board switch, it would be physically impossible to reset it manually on each instruction.

**Reduction of States 19/21 and 20/22**

States 19 and 21 are identical because they issue the same control word and have the same next-state logic, considering that states 20 and 22 also have identical control words and next-stage logic. Only the datapath evaluates the different conditions in states 19 and 21. The FSM itself does not distinguish between the two. It merely orders the datapath to evaluate a condition (either condition) and write it to the appropriate flip-flops. Similarly, states 20 and 22 rely on different conditions, but the conditions can be evaluated in unity as COND<1> OR COND<0> in order to determine whether or not to issue the control word. Again, the control unit simply orders the loading of the accumulator, the gating of X, and the selection of arithmetic without distinguishing between which condition specifically is true as long as one of them is true.

**An Explanation of the ALU**

The PASS and NOT instructions operate on the B input, so only AND, SUB, INC, or DEC will activate the A input. The INC and DEC instructions operate on the A input, so these two signals deactivate the B input. The SUB and NOT instructions invert each bit in B, so these two signals are XORed with B. SUB requires a negative 2's complement representation of B in order to facilitate A+(-B), so in addition to inverting its bits, it inputs a carry-in (c0) to add 1 to the LSB in the chain of full-adders. Other signals that require this carry-in are INC and DEC. DEC is the addition of -1, so it also requires all carry-in bits from 0 to 7, which effectively inputs 1111111 or -1 in 8-bit 2's complement as the second input to the full-adder. The logic-extender gates either zeros or A "OP" B into an AND gate that outputs to the datapath, where "OP" represents either AND, OR or the output of the full-adders. The ellipses on either end of the logic-extender show that the logic depicted only represents a single cell of the entire logic-extender, where subscript *i* represents the index of the cell. The final carry bit of the full-adders represents the overflow condition.