**Lab 12: Control Unit Construction**

**Primary Objectives**

1. Design control unit for microprocessor

2. Implement control unit

3. Test control unit

*Objective 1 Design*

To design the control unit for the microprocessor, I split the control unit into several parts based on the schematic provided in the lecture slides. I started by constructing a CAR to load the addresses into the Control Store ROM. This was done by setting up a counter to keep track of what address the program was currently running and connecting it to the LOAD and RESET signals. I then put together the data signal logic which just used a two-input mux to determine whether to load the NEXT\_ADDRESS or OPCODE value into the CAR based on whether the previous program had finished running. I then set up the Operand Select by using a mux to decide which operand constant to output based on the 3-bit BA value. The last major element of the control unit that I implemented was the load signal logic. This output a value for LOAD dependent on if the current CAR address was x00 or the mode was set to test a PZN condition. The rest of the control unit logic consisted of connecting the elements’ inputs and outputs and separating the INSTRUCTION input and CTRL WORD output into their separate values to allow for simple processing between the control unit and datapath.

*Objective 2 Implementation*

Below are screenshots of each of the different elements of the control unit, including the complete control unit itself.

A diagram of a computer

Description automatically generated

Control Unit Implementation

A computer screen shot of a circuit

Description automatically generated

CAR Implementation

A diagram of a circuit

Description automatically generated

Data Signal Logic Implementation

A diagram of a circuit

Description automatically generated

Operand Select Implementation

A diagram of a circuit

Description automatically generated

Load Signal Logic Implementation

Below is the overall implementation of the final microprocessor. The lower box is the control unit, the upper box is the datapath, and the upper right box is the RAM.

A diagram of a computer

Description automatically generated

Final Microprocessor Implementation

The final portion of the implementation consisted of programming the Control Store ROM. The table below contains all the opcode required to be implemented in the microprocessor. The final opcode (xB) is the one requested by the TA. This is represented by the following expression:

if ( M[A] + M[C] > 4 )

M[A] ← A

else

M[C] ← C

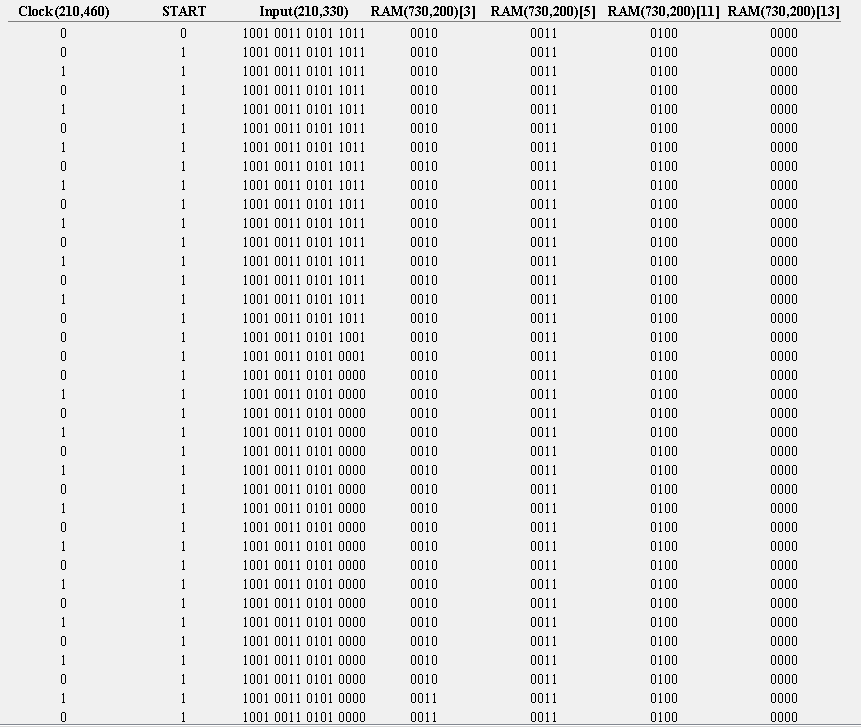
|  |  |  |  |
| --- | --- | --- | --- |
| **ROM ADDRESS** | **RTL** | **BINARY** | **HEX** |
| 00 | CAR < x00 | 10 0111 0000 0000 0000 | x27000 |
| 10-13 | R0 < CONST(OP A) R1 < CONST(OP B)  ADDR OUT < R0, DATA OUT < R1  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  01 0000 0010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x10200  x27000 |
| 20-24 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < M[R1]  M[R0] < R2  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0010 0001 0000 0101  01 0000 0100 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x02105  x10400  x27000 |
| 30-37 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R3 < M[R1]  R4 < M[R2]  R5 < R3 + R4  M[R0] < R5  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0010 0001 0000 0111  00 0100 0001 0000 1001  00 0110 1000 0100 1011  01 0000 1010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x02107  x04109  x0684B  x10A00  x27000 |
| 40-47 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R3 < M[R1]  R4 < M[R2]  R5 < R3 - R4  M[R0] < R5  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0010 0001 0000 0111  00 0100 0001 0000 1001  00 0110 1000 0101 1011  01 0000 1010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x02107  x04109  x0685B  x10A00  x27000 |
| 50-57 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R3 < M[R1]  R4 < M[R2]  R5 < R3 \* R4  M[R0] < R5  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0010 0001 0000 0111  00 0100 0001 0000 1001  00 0110 1000 0110 1011  01 0000 1010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x02107  x04109  x0686B  x10A00  x27000 |
| 60-65 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < M[R1]  R3 < R2 >> 1  M[R0] < R3  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0010 0001 0000 0101  00 0100 0000 1100 0111  01 0000 0110 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x02105  x040C7  x10600  x27000 |
| 70-75 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < M[R1]  R3 < NOT(R2)  M[R0] < R3  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0010 0001 0000 0101  00 0100 0000 1000 0111  01 0000 0110 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x02105  x04087  x10600  x27000 |
| 80-87 | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R3 < M[R1]  R4 < M[R2]  R5 < R3 AND R4  M[R0] < R5  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0010 0001 0000 0111  00 0100 0001 0000 1001  00 0110 1000 1001 1011  01 0000 1010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x02107  x04109  x0689B  x10A00  x27000 |
| 90-9F | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R3 < M[R2]  R3 < R3  P+N: CAR < x00  R1 < M[R1]  M[R0] < R1  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0100 0001 0000 0111  00 0110 0000 0010 0111  10 0101 0000 0000 0000  00 0010 0001 0000 0011  01 0000 0010 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x04107  x06027  x25000  x02103  x10200  x27000 |
| A0-AF | R0 < CONST(OP A) R1 < CONST(OP B)  R2 < CONST(OP C)  R1 < M[R1]  R2 < M[R2]  R3 < R2-R1  Z: CAR < xAA  R4 < 1  M[R0] < R4  CAR < x00  R4 < 0  M[R0] < R4  CAR x00 | 00 0001 0010 0011 0001  00 0001 0100 0011 0011  00 0001 0110 0011 0101  00 0010 0001 0000 0011  00 0100 0001 0000 0101  00 0100 0010 0101 0111  10 0010 1010 1001 0000  00 0000 0000 0001 1001  01 0000 1000 0000 0000  10 0111 0000 0000 0000  00 0000 0000 0000 1001  01 0000 1000 0000 0000  10 0111 0000 0000 0000 | x01231  x01433  x01635  x02103  x04105  x04257  x22AA0 x00015  x10800  x27000  x00005  x10800  x27000 |
| B0-BD | R0 < CONST(OP A) R1 < CONST(OP C)  R2 < M[R0]  R3 < M[R1]  R4 < R2 + R3  R5 < 1  R5 < R5<<1  R5 < R5<<1  R6 < R4-R5  N+Z: CAR < xBC  M[R0] < R0  CAR < x00  M[R1] < R1  CAR < x00 | 00 0001 0010 0011 0001  00 0001 0110 0011 0011  00 0000 0001 0000 0101  00 0010 0001 0000 0111  00 0100 0110 0100 1001  00 0000 0000 0001 1011  00 1010 0000 1101 1011  00 1010 0000 1101 1011  00 1000 1010 0101 1101  10 0011 1011 1100 0000  01 0000 0000 0000 0000  10 0111 0000 0000 0000  01 0010 0010 0000 0000  10 0111 0000 0000 0000 | x01231  x01633  x00105  x02107  x04649  x0001B  x0A0DB  x0A0DB  x08A5D  x23BC0  x10000  x27000  x12200  x27000 |

(All unused ROM addresses contain x00000.)

*Objective 3 Testing*

For this system, I tested each operation on a series of random values in the RAM after writing the control words to the ROM. To demonstrate a particular case, I will walk through how I wrote the instructions (shown above) for OPCODE x9. OPCODE x9 sets M[B] to M[A} iff M[C] = 0. The first three instructions I wrote simply set the A, B, and C constants to registers R0, R1, and R2, respectively. I then copied M[R2] to R3. This allowed me to hold the value of M[C] for testing its value against 0. The next instruction set MODE to 1 and tested the condition P+N. If the condition was true, then the next address loaded was the IDLE address x00. This essentially quits the operation if the value in R3, which holds M[C], does not equal 0. Otherwise, the program moves on and sets R1 as M[R1], allowing it to hold the value of M[B]. The second-to-last instruction then takes the value in R1 and copies it to M[R0], which finally sets M[A] to the value of M[B]. The program then runs instruction x27000, which is the default instruction to move the program to IDLE address x00.

Below is a log which demonstrates the functionality of OPCODE x9 on the microprocessor. Note how, when OP C is set to address xB (11) which is non-zero, all the RAM values remain the same. When OPCODE x9 is run again with OP C set to address x0, an address containing value 0, the value of M[A] (address x3) is replaced with the value of M[B] (address x5).



OPCODE x9 Test Log

**Conclusion**

This lab was quite difficult. I initially struggled with creating the data and load signal logic, but after enough iterations, I came to a solution that worked quite well. The rest of the smaller logic in the control unit was rather simple and easy to implement. Writing the OPCODE instructions was not terribly difficult, especially as I got more comfortable writing them with the later ones, but it was easy to write one register wrong or convert an instruction to hex incorrectly. This made troubleshooting particularly tedious. Overall, it was a good learning experience though.