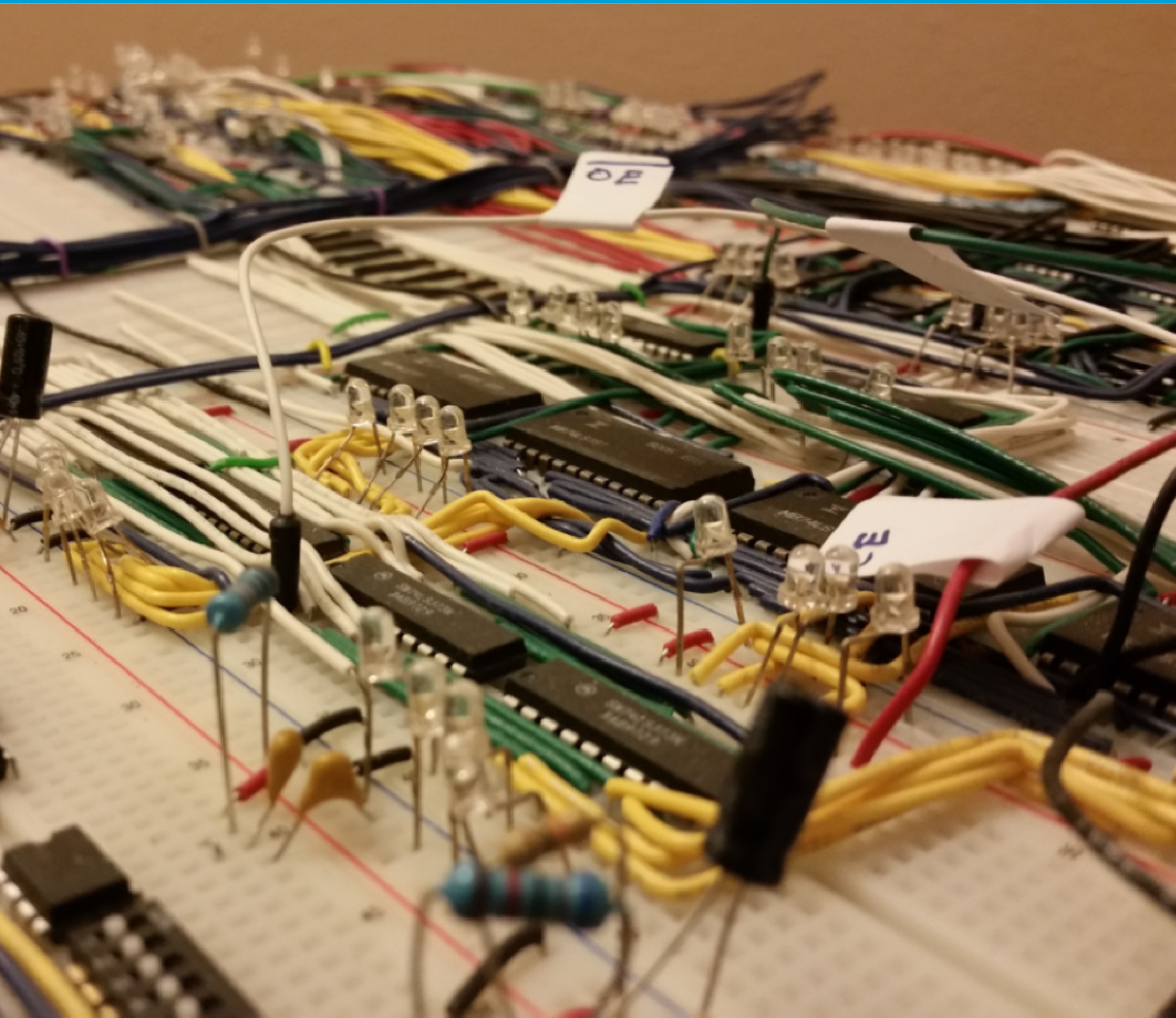
Kenneth Hall

16-BIT BREADBOARD COMPUTER



16-Bit Breadboard Computer Report

Prepared By

Kenneth Hall

Spring 2018

Austin, TX

CONTENTS

TABLES

FIGURES

EXECUTIVE SUMMARY

This report details the planning, design, and implementation of creating the 16-bit Breadboard Computer. Building the computer was a personal project started in Summer 2017 and will be completed Summer 2018. It was inspired by a Simple as Possible (SAC) computer from Ben Eater. The goal of the project was to improve personal engineering skills by building a computer that could manipulate memory, perform logical and mathematical operations, and use control flow instructions. Building the computer provided an opportunity to research, design, model, and troubleshoot computer architecture models. Designing the computer included weighing the pros and cons of different microarchitectural. For example, lower cost might be chosen over speed, or simple and concise operating modes were preferred to complex and robust. This project provided the opportunity to develop and improve good engineering practices skills outside of the classroom.

This goal of this project was to gain first-hand experience with electrical engineering methodologies such as researching, designing, and organizing architecture and layout. Additionally, this included troubleshooting, simulating, and measuring performance or response. The computer was built at a low enough level to demonstrate concepts and avoided schemes that were tedious or repetitive. For example, discrete arithmetic logic units (ALU) were used rather than adders built from logic gates. Another example includes utilizing a non-pipelined design; this was chosen since the computer did not need to be fast and building multiple copies of modules would have greatly increased time and cost.

Modules were planned and built to meet the project's solution specifications to be informative. The modules were: system clock, bus, program counter (PC), memory, register file, ALU, and a user interface. The modules to be completed are the control store and instruction register. The system clock supported multiple speeds that show what the current instruction is doing or allow the computer to be stepped through by hand. The program counter supports setting and resetting. Memory access did not need to be fast which minimized overhead. The ALU supports basic operations, such as NAND or addition, but has allocations for implementing for future features. Lastly, the register file should be large enough for typical programs and minimize cost. Overall, modules were should be functionally complete and cost efficient.

Implementing each module required designing schematics that fit the solution requirements to be functionally complete. This required researching parts, implementing features, and determining how the module will be controlled. For example, building the PC circuit required choosing between low-overhead latches in combination with the ALU versus a higher overhead dedicated counting latch. Implementing the circuit required planning for what modes and operations would be supported, such as allowing the PC to be set for branches. Lastly, writing the microinstructions for how the control store would operate the module was completed. This process was repeated for each module.

Building the 16-bit computer was a difficult but rewarding experience and provided the opportunity to define and maintain a long-term project. The project improved my technical skills and reinforced classroom information. Although the project was outside of my comfort-zone and required patience, I learned from my mistakes and development became easier overtime.

1.0 INTRODUCTION

This report explains the goal, solution, and implementation of the 16-bit computer. The next section details what the final intention of the project is and how building a computer will help meet those objectives. Next, the report describes what properties and instructions the computer needed to meet the projects intention and explains module features. Lastly, implementation of the computer is described. This includes how each circuit was built, what components were chosen, and the operating modes. Building and developing a Turing Complete computer was an engaging opportunity that provided hands-on engineering experience for new principles.

2.0 DESIGN PROBLEM

The goal of this project was to design and build a 16-bit computer. Specifically: "…a mechanism that does two things: It directs the processing of information and it performs the actual processing of information" [1]. This computer would be able to store and retrieve data from memory, manipulate that data, and execute a program from a well-defined instruction set. The computer does not need to have an extensive set of instructions, such as multiplying or calculating a square root, but the capabilities should demonstrate computer architecture concepts.

To demonstrate the architecture, the computer should be concise, straight-forward, and have an intuitive layout. The computer should be able to operate such that micro-instructions can be clearly seen. Indicators, such as LEDs or seven-segment displays, should where data is being moved or manipulated. The user should be able to "look into" registers and check the computer's state.

The computer should at a minimum be able to: store and load data, perform simple mathematical and logical operations, and use control flow instructions. Working with 16-bit words should be prioritized over byte operand instructions. Completing memory access instructions should be prioritized. Typically, the modules are not completed simultaneously, so there is possibility to reduce hardware cost by sharing resources. For example, using binary counter ICs could be substituted with registers and using the ALU to increment.

3.0 DESIGN SOLUTION

This section explains what solution was chosen for each module and their benefits or drawbacks. The goal of the clock, reset, and debug modules was to provide an access point for operating the computer. The clock should be able to manually stop, started, and resumed. The user should be able to see the computer's state cycle by cycle. The clock should have an operating mode switch and step-through capabilities through a button. Additionally, the clock should run autonomously at multiple speeds. However, until the computer is completed, a slower frequency is adequate for troubleshooting. Lastly, a reset button should be implemented that will reset system registers to expedite troubleshooting.

The program counter should contain the address of the next instruction to be executed. Therefore, the module should be able to easily increment its data. Additionally, the register will need to be set to allow for branch instructions.

Memory should emphasize simplicity and reduce overhead. Speed is not a priority and word length instructions will be accommodated. It should be byte addressable to reduce backtracking for future features. Memory should have enough capacity to store a user program and have enough system space for future vector tables. Program length should accommodate the requirements to highlight features of the computer. This typical program might load a value, load another value, add them, branch if even or odd, and store them accordingly. Additionally, the mechanism used for memory does not need to be fast. Many of the ICs used have an execution time on the order of $100\ \mu s$ or less, but the computer will typically be stepped through by hand.

The register file should have enough capacity to facilitate user programs. Any registers used in hardware will not be accessible by the user and can be omitted from this module. Cost and lower overhead should be chosen over speed.

The arithmetic logic unit (ALU) should prioritize frequently used operations. ADD and AND operations account for 14% of 80x86 instructions used in a typical program [2], therefore they were the first to be implemented. Richer instructions should be implemented in the future, but aim to minimize complexity.

4.0 DESIGN IMPLEMENTATION

This section explains how each completed module was developed. Truth tables, components, and IC operating modes are shown here. In the following diagrams, filled arrowheads represent data movement and open arrow heads represent control lines. Lines are abbreviated with a slash and a number, such as 16 lines to the bus. Some signals are omitted, such as LEDs, their intermediate registers, or V_{CC}. Since it is not possible to "look into" an IC directly, a second register is sometimes added that latches a value for the LED.

In each photo, a black wire is used for ground and red for V_{CC}. White represents 16-bits of data usually to and from the bus. Red and black are sometimes used for this signal. Blue show control signals. Yellow and green are reserved for signals within the module. Red sometimes carries this signal.

The completed modules are: bus, clock, debug, program counter, MDR, MAR, register file, and arithmetic logic unit. The modules in work are the control store, instruction register, and pre-processor memory writer. Additionally, an Arduino based programmer was written to flash CAT28C16A EEPROMs. Since the control store is not complete, the control store EEPROMs contain temporary microinstructions. Figure 1 through Figure 4 show the full computer.

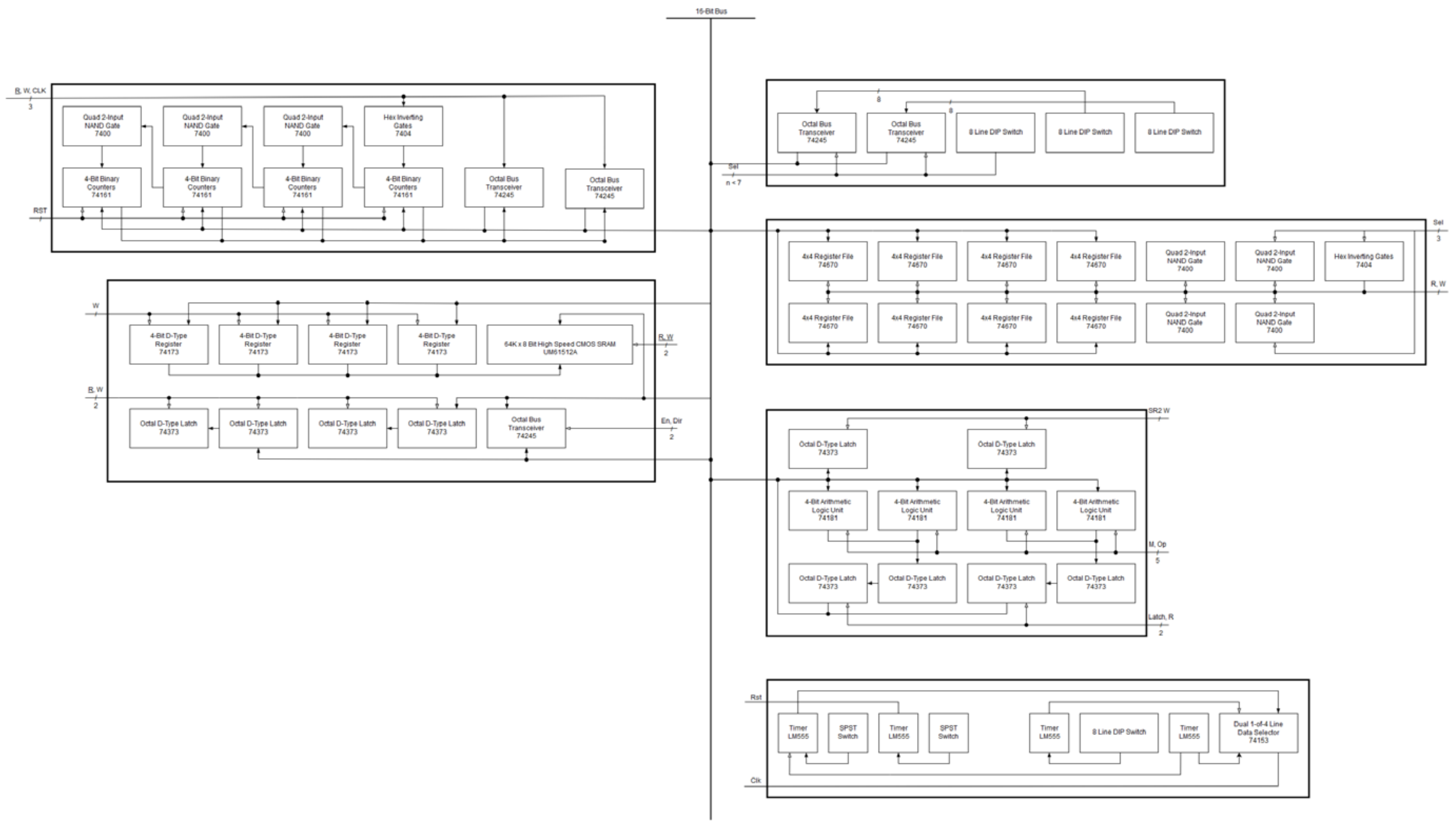

16-Bit Bus
R, W, CLK
3
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
Hex Inverting Gates 7404
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
Octal Bus Transceiver 74245
Octal Bus Transceiver 74245
RST
W
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
64K x 8 Bit High Speed CMOS SRAM UM61512A
R, W
2
R, W
2
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal Bus Transceiver 74245
En, Dir
2
8
8
Octal Bus Transceiver 74245
Octal Bus Transceiver 74245
8 Line DIP Switch
8 Line DIP Switch
8 Line DIP Switch
Sel
n < 7
Sel
3
4x4 Register File 74670
4x4 Register File 74670
4x4 Register File 74670
4x4 Register File 74670
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
Hex Inverting Gates 7404
R, W
4x4 Register File 74670
4x4 Register File 74670
4x4 Register File 74670
4x4 Register File 74670
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
SR2 W
Octal D-Type Latch 74373
Octal D-Type Latch 74373
4-Bit Arithmetic Logic Unit 74181
4-Bit Arithmetic Logic Unit 74181
4-Bit Arithmetic Logic Unit 74181
4-Bit Arithmetic Logic Unit 74181
M, Op
5
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Latch, R
2
Rst
Timer LM555
SPST Switch
Timer LM555
SPST Switch
Timer LM555
8 Line DIP Switch
Timer LM555
Dual 1-of-4 Line Data Selector 74153
Clk

Figure 1. 16-bit computer layout.

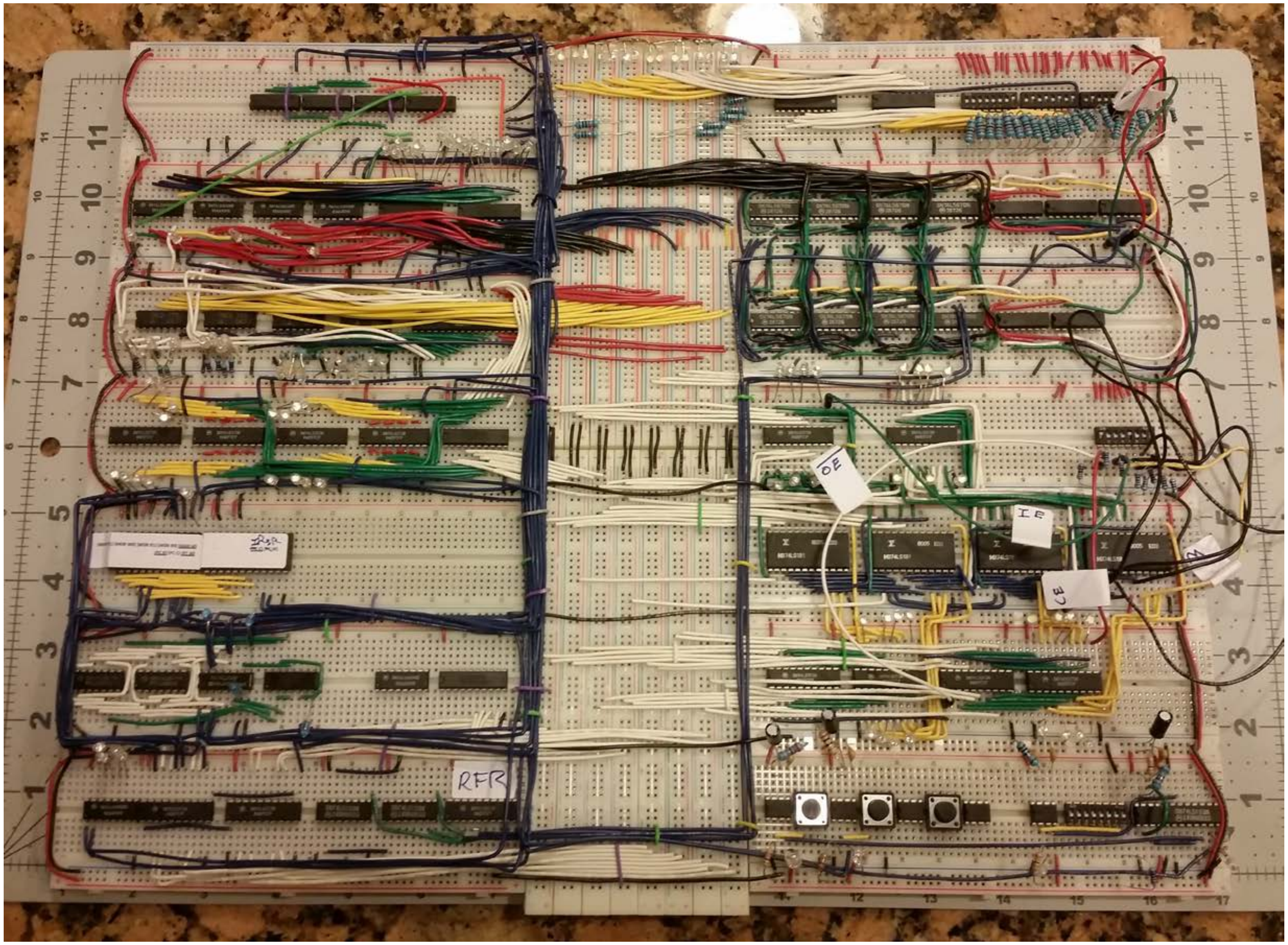


Figure 2. 16-bit computer, top-down.

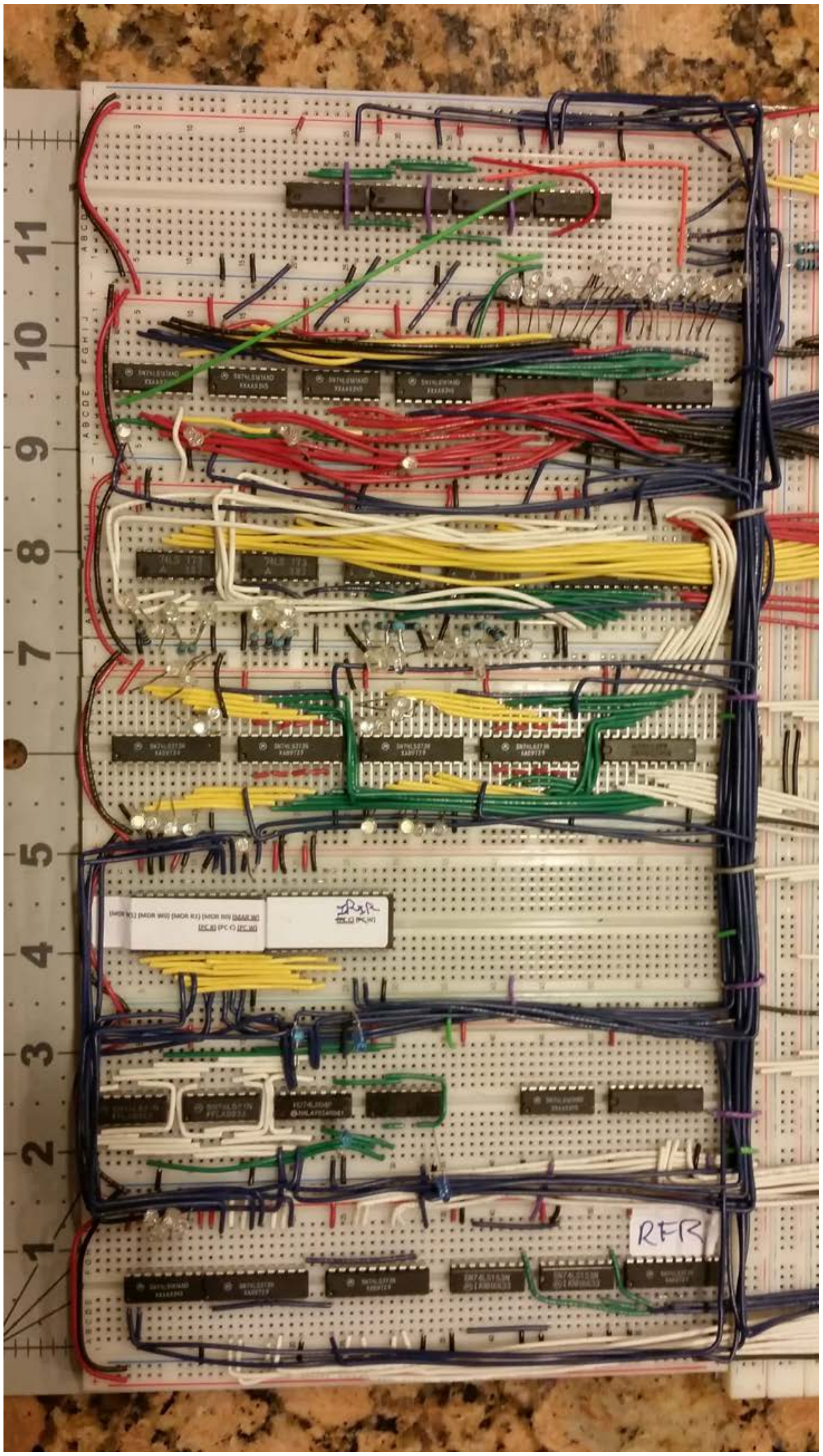


Figure 3. 16-bit computer, top-down, left.

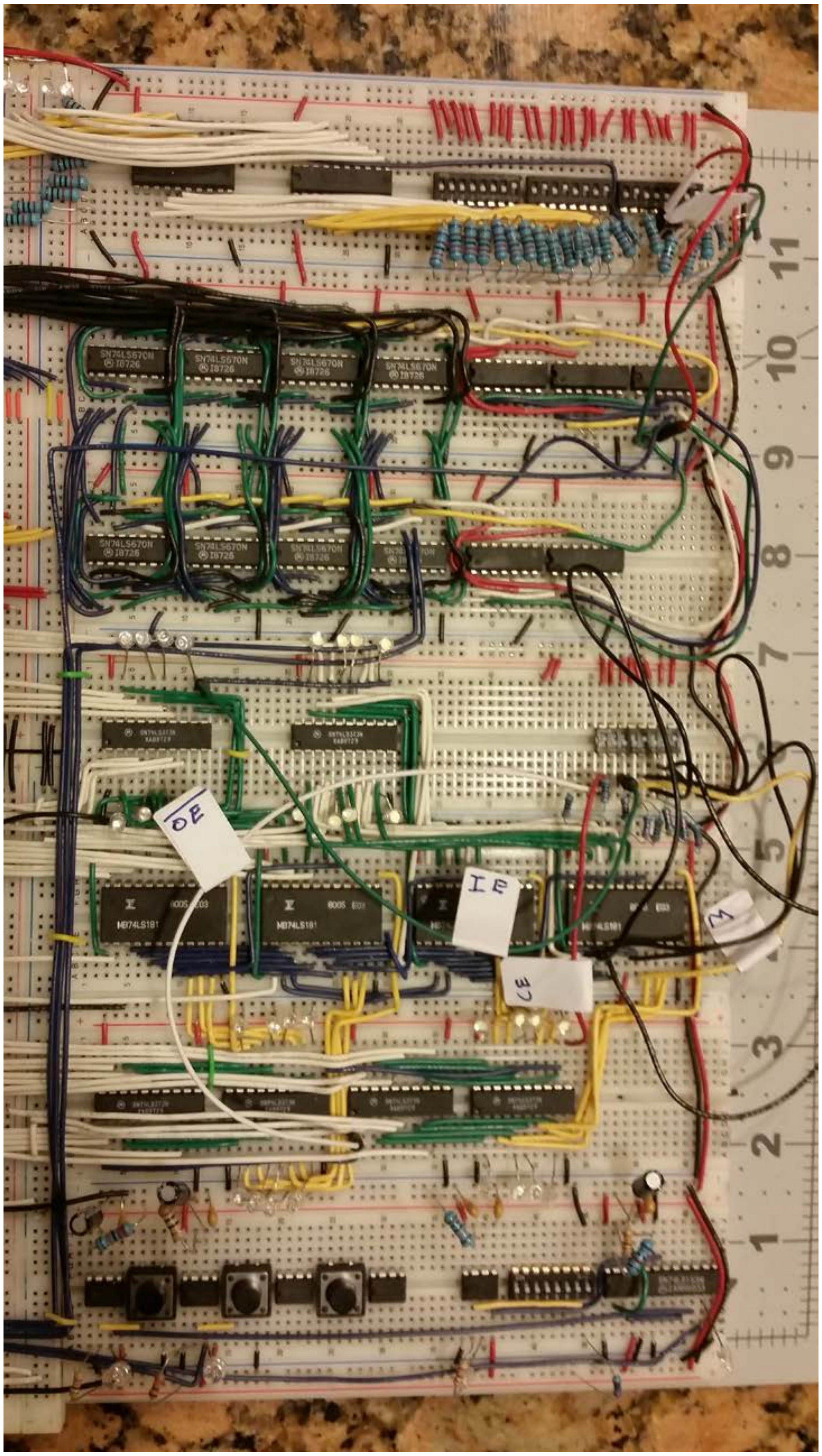


Figure 4. 16-bit computer, top-down, right.

4.1 Bus

The bus transports all non-control signals for the computer. It is made of 32 breadboard terminal strips, 16 LEDs, and 16 resistors. The terminal strips are divided into 16 pairs arranged vertically. Each LED corresponds to one bit out of 2-byte words. When the LED is on, that bit is high.

4.2 Clock and Reset

This module controls the clock frequency, including manually stepping through, and a reset switch. The system clock consists of five LM555 timers, one multiplexor 74LS153, two SPST switches, and one 8-line DIP switch. The configuration is shown in Figure 5 and Figure 6.

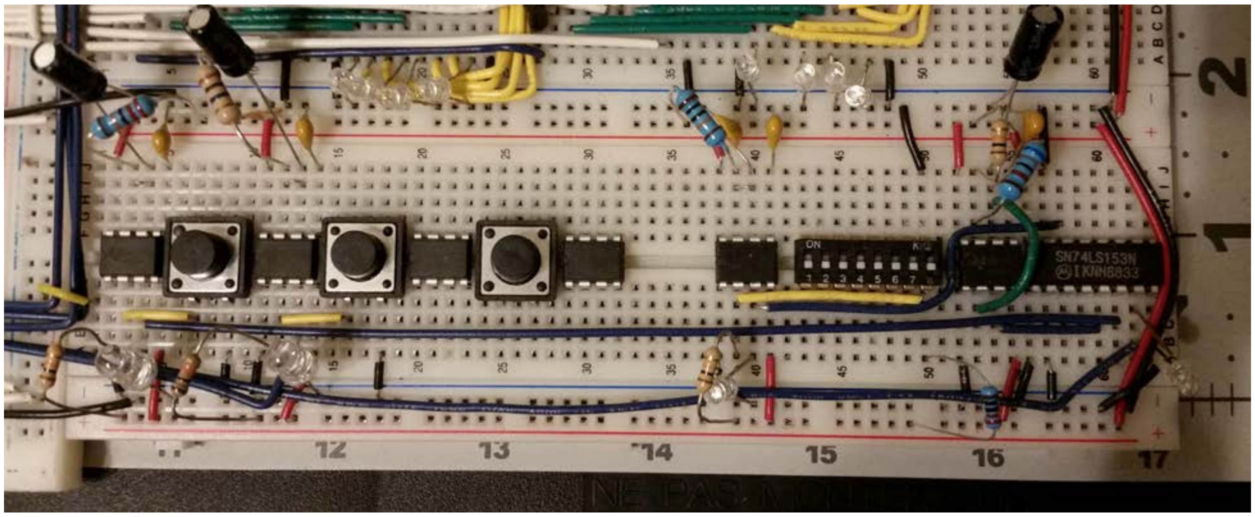


Figure 5. System clock, top-down.

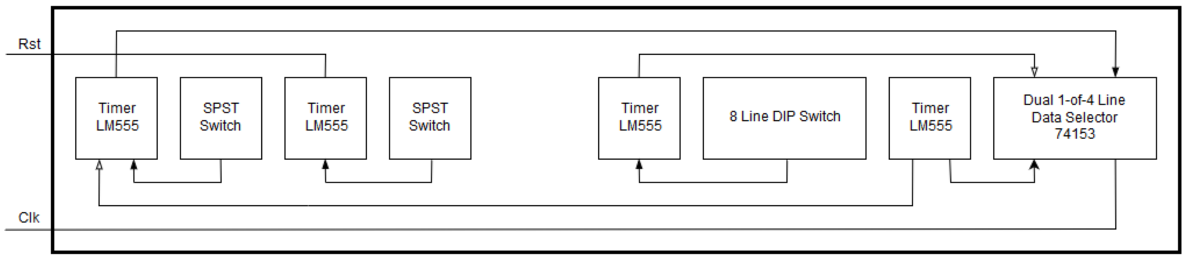


Figure 6. System clock layout.

The 8-line DIP switch selects one of two modes: a 1 Hz 50% duty square wave or a debounced SPST switch to step through clock cycles. The DIP switch implements only one of eight signals with room for future features. Similarly, the multiplexor implements two of four lines. For larger programs in the future, an operating mode might be implemented on the order of 100 Hz. Another option is to use the cycle time of the slowest IC. Additionally, the circuit also includes a master reset switch to set PC and other modules to zero. All SPST and DIP switches in this circuit are debounced with LM555s; these components were on-hand and ready to use. In the future, a simpler RC circuit will be used to reduce overall cost.

4.3 Debug

The debugging module is used to force values onto the bus; it contains two octal bus transceiver 74LS245 and three 8-line DIP switches. The configuration is shown in Figure 7 and Figure 8.

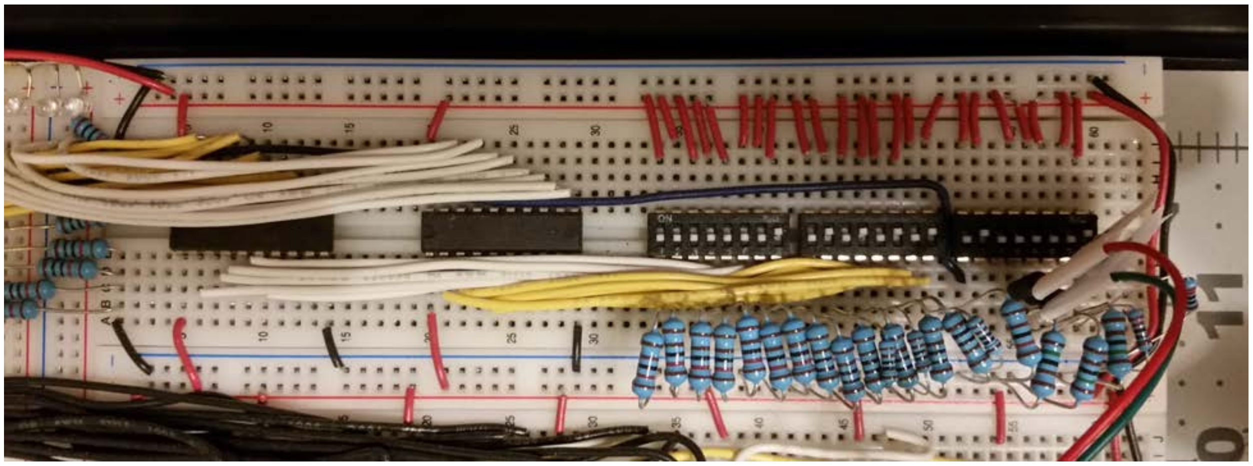


Figure 7. Debug, top-down.

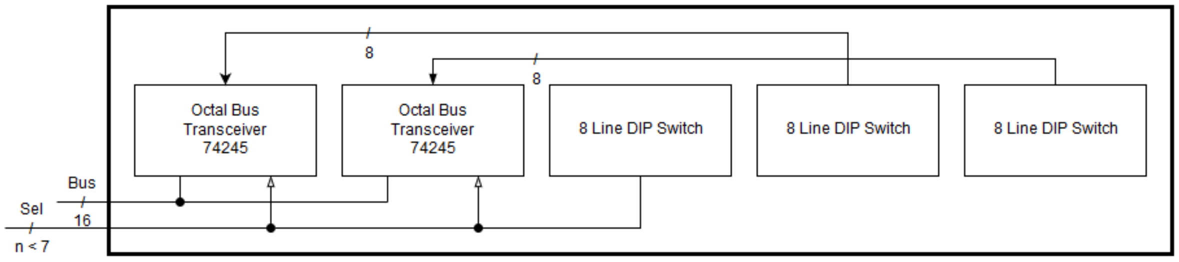


Figure 8. Debug layout.

Two switches select what bit is to be written to the bus. The last switch enables the two octal bus transceivers—the transceivers are set only one direction and provide a high impedance state when the module is not in use. The unused positions in this switch are used for dynamic testing and debugging while building new modules.

4.4 Program Counter

The program counter (PC) contains the address of the next instruction to be processed. The program counter will need to be reset prior to the first instruction and is able to be set to a certain address to allow for branch instructions. It consists of four 4-bit binary counters, two 8-bit tristate buffers, three quad 2-input NAND gates, and hex inverting gates. The configuration is shown in Figure 9 and Figure 10.

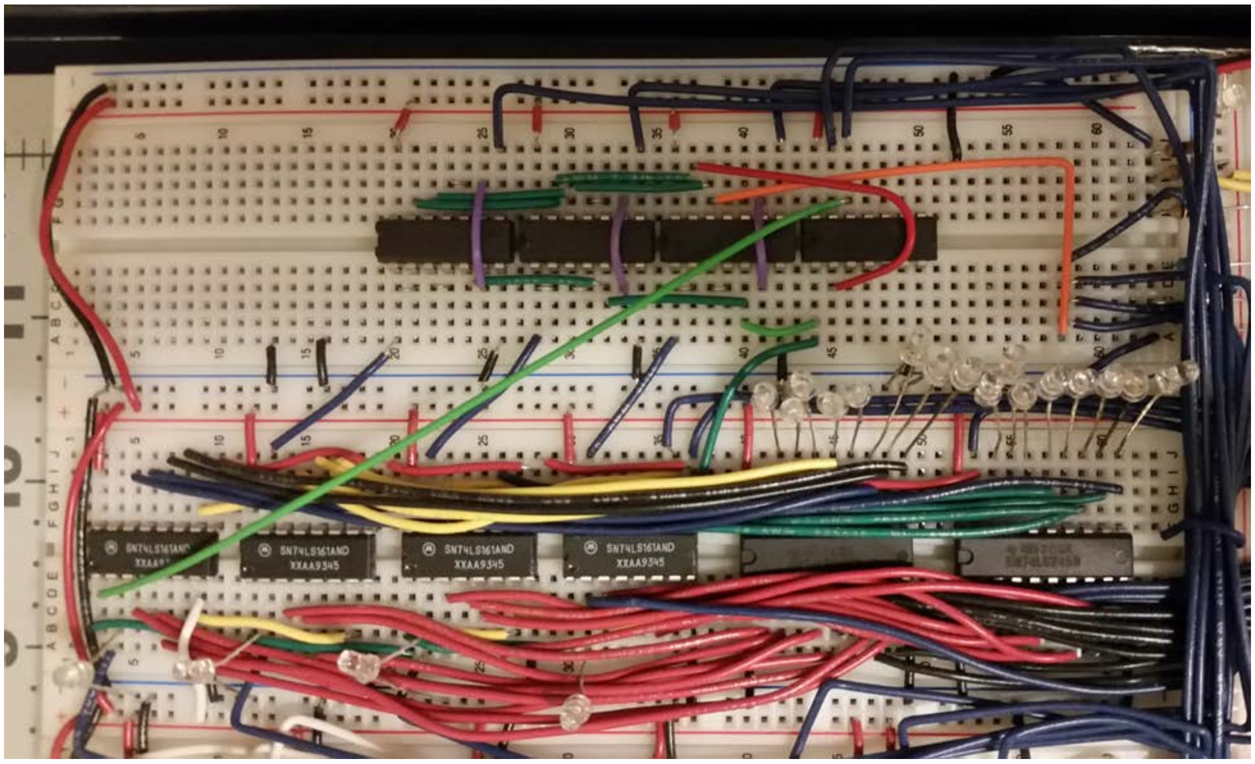
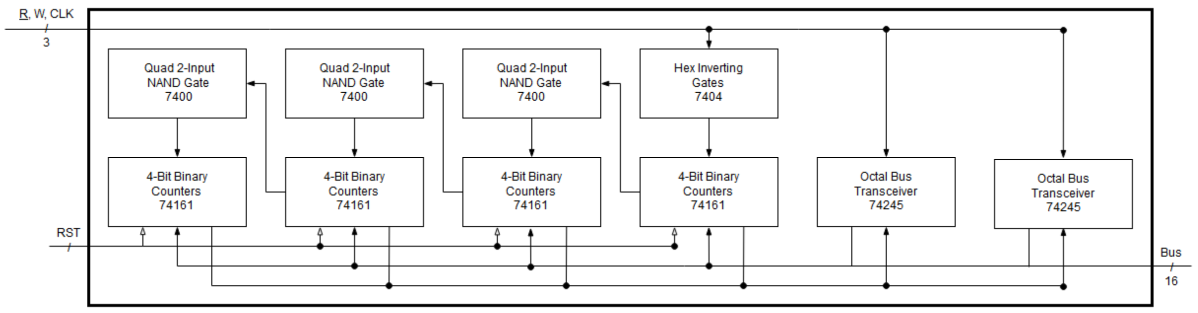


Figure 9. Program counter, top-down.



R, W, CLK
3
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
Quad 2-Input NAND Gate 7400
Hex Inverting Gates 7404
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
4-Bit Binary Counters 74161
Octal Bus Transceiver 74245
Octal Bus Transceiver 74245
RST
Bus
16

Figure 10 Program counter layout.

The 74LS161 binary counter increments the address based on the computer's state. All output bits were connected to 74LS245 buffers to prevent overwriting the bus. The active low master reset (MR) signal was connect to a master reset SPST switch. The clock bit, CP, was connected to a decoder for testing but will be controlled by the control store in the future. Since four binary counters were connected in series, the first counter would use the control store signal for normal operation, and the remaining counters would use the previous counter's terminal count output bit (TC). This bit signaled when a counter had counted its' full range. However, resetting the counters required a clock pulse. Therefore, the counters would need their respective clock or TC signal at certain times, and need to receive a coordinated pulse to reset. This logic was reduced with LogicAid and then converted to a NAND gate equivalent to reduce cost. The 74LS00 quad-NAND gates and the 74LS04 inverter were used to create this multiplexer.

The 74LS245 octal bus transceiver was used to buffer the program counter's output. Since the buffer was used for outputting to the bus only, the direction bit (DIR) was always set in that direction. The enable bit (E), was used to control when the PC sends data to the bus. The remaining lines consist of input to output pairs.

4.5 Memory

The memory module contains the memory address register (MAR), memory data register (MDR), and physical memory. The MDR contains the contents of the address in MAR. It is made from one 64K x 8 CMOS Sram UM61512A, four 4-bit D-Type Register 74LS173, four octal D-Type latch 74LS373, one octal bus transceiver 74LS245, two dual 4-input AND gate

74LS21, one hex inverting gate 74LS04, and one quad 2-input OR gate 74LS32. The configuration is shown in Figure 11 and Figure 12.

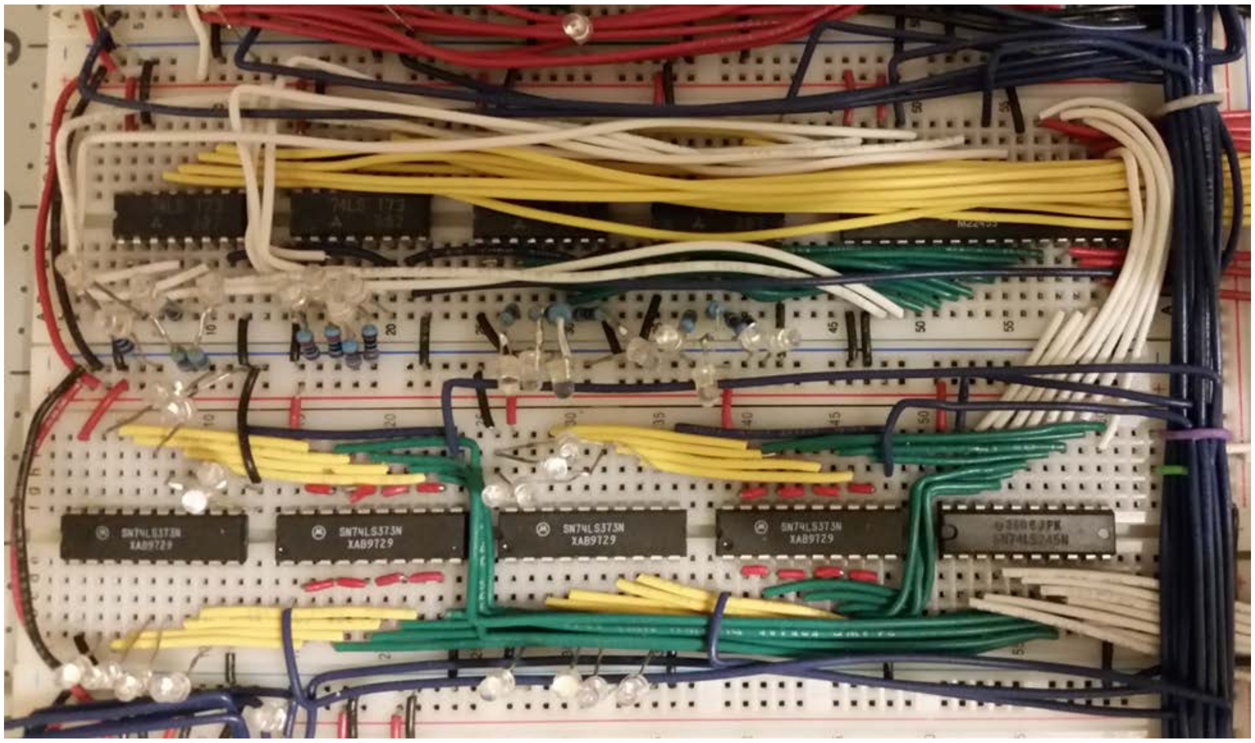


Figure 11. Memory, top-down.

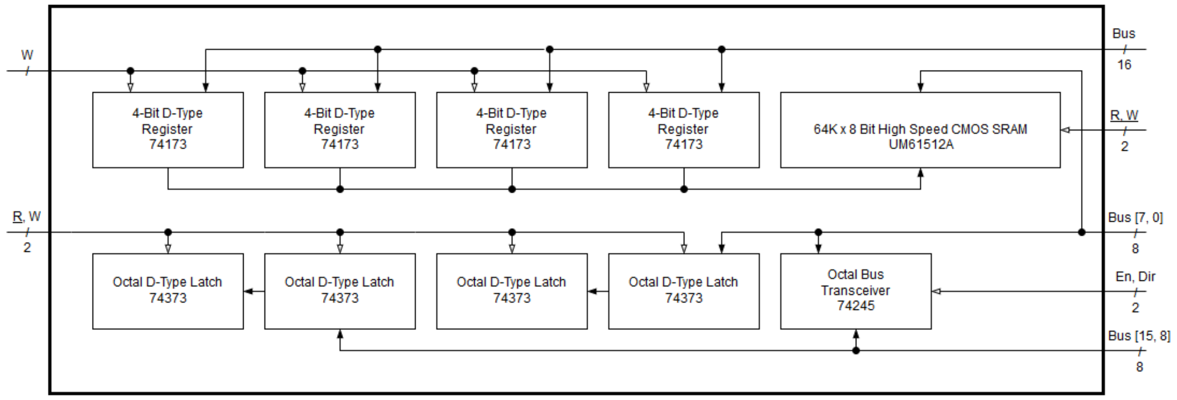

W
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
4-Bit D-Type Register 74173
64K x 8 Bit High Speed CMOS SRAM UM61512A
Bus
16
R, W
2
R, W
2
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal D-Type Latch 74373
Octal Bus Transceiver 74245
Bus [7, 0]
8
En, Dir
2
Bus [15, 8]
8

Figure 12. Memory layout.

The MAR received a 16-bit address from the bus. This value determined what place in memory was to be accessed. A write enable bit, controlled by the control store, would then allow an address to be sent to memory. The MDR contained four octal D-type latches. However, the latches were connected in series and held identical values—this allow LEDs to be connected to see what values the latch pairs held. Since memory was byte addressable, the high and low bytes of the bus had to be both accessible by the MDR. The octal bus transceiver 74LS245 was configured to allow the high byte to be copied onto the lower byte on the bus. This data path is shown in Figure 13 in red.

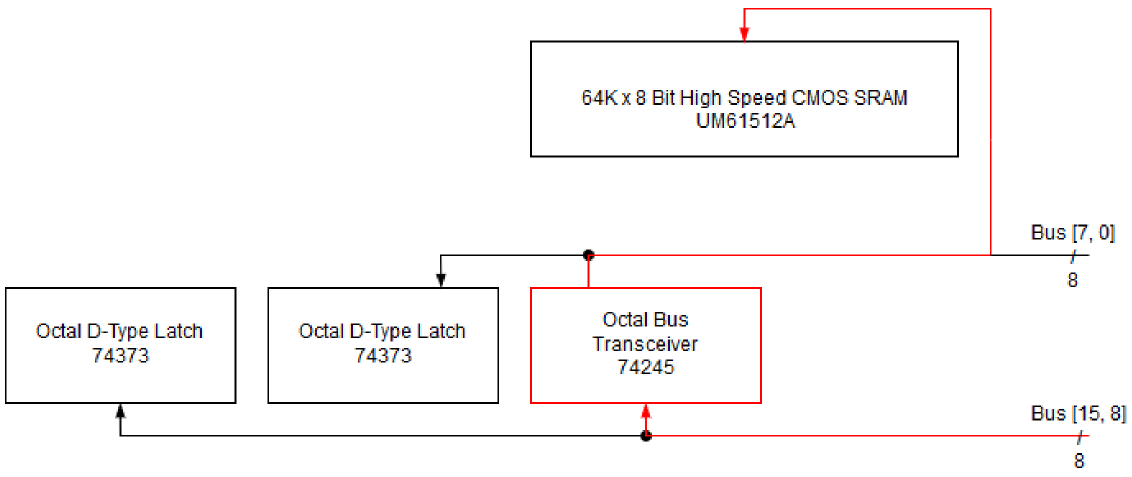


Figure 13. Memory high to low byte data path.

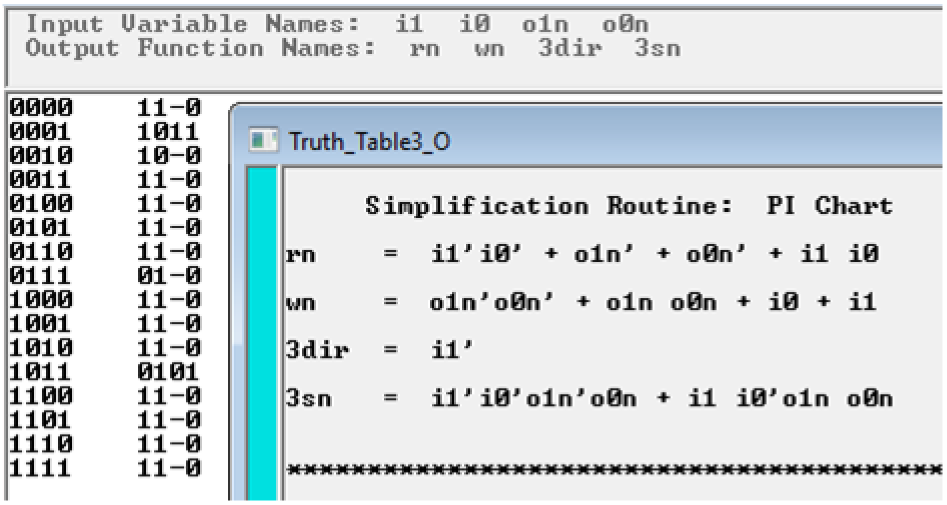
At this point, the control store must keep track of the following MDR signals: read byte 1, read byte 0, write byte 1, write byte 0; the signals to write to memory are: memory read, memory write, tri-state buffer enable, tri-state buffer direction. However, only certain combinations of these signals are ever used; the truth table, shown in Table 1, shows all possible states.

Table 1. Memory logic truth table

W1	W0	R1	R0	R mem	W mem	3s dir	3s en	State
0	0	0	0	1	1	-	0	MDR to bus
0	0	0	1	1	0	1	1	r1 to mem
0	0	1	0	1	0	-	0	r0 to mem
0	0	1	1	1	1	-	0	Default (off)
0	1	0	0	1	1	-	0	
0	1	0	1	1	1	-	0	

0	1	1	0	1	1	-	0	
0	1	1	1	0	1	-	0	Mem to R0
1	0	0	0	1	1	-	0	
1	0	0	1	1	1	-	0	
1	0	1	0	1	1	-	0	
1	0	1	1	0	1	0	1	Mem to R1
1	1	0	0	1	1	-	0	
1	1	0	1	1	1	-	0	
1	1	1	0	1	1	-	0	
1	1	1	1	1	1	-	0	Bus to MDR

Since the memory and tri-state control signals are dependent on what byte is being written or read, the number of control lines can be reduced from eight to four. The table was entered into logic aid to find a minimally reduced solution. The LogicAid simplification routine is show in Figure 14.



```
Input Variable Names:  i1  i0  o1n  o0n
Output Function Names:  rn  wn  3dir  3sn

0000    11-0
0001    1011
0010    10-0
0011    11-0
0100    11-0
0101    11-0
0110    11-0
0111    01-0
1000    11-0
1001    11-0
1010    11-0
1011    0101
1100    11-0
1101    11-0
1110    11-0
1111    11-0
```

```
Simplification Routine:  PI Chart

rn    =  i1'i0' + o1n' + o0n' + i1 i0

wn    =  o1n'o0n' + o1n o0n + i0 + i1

3dir  =  i1'

3sn   =  i1'i0'o1n'o0n + i1 i0'o1n o0n

*****************************************
```

Figure 14. Memory logic simplification.

Lastly, these equations will be implemented within the control store using CAT28C16A EEPROMs. These are shown in Figure 3 below the memory module.

4.6 Register File

The register file provides eight 16-bit registers for the user. It is made from eight 4x4 register file 74LS670, four Quad 2-Input NAND gate 74LS00, and a hex inverter 74LS04. The configuration is shown in Figure 15 and Figure 16.

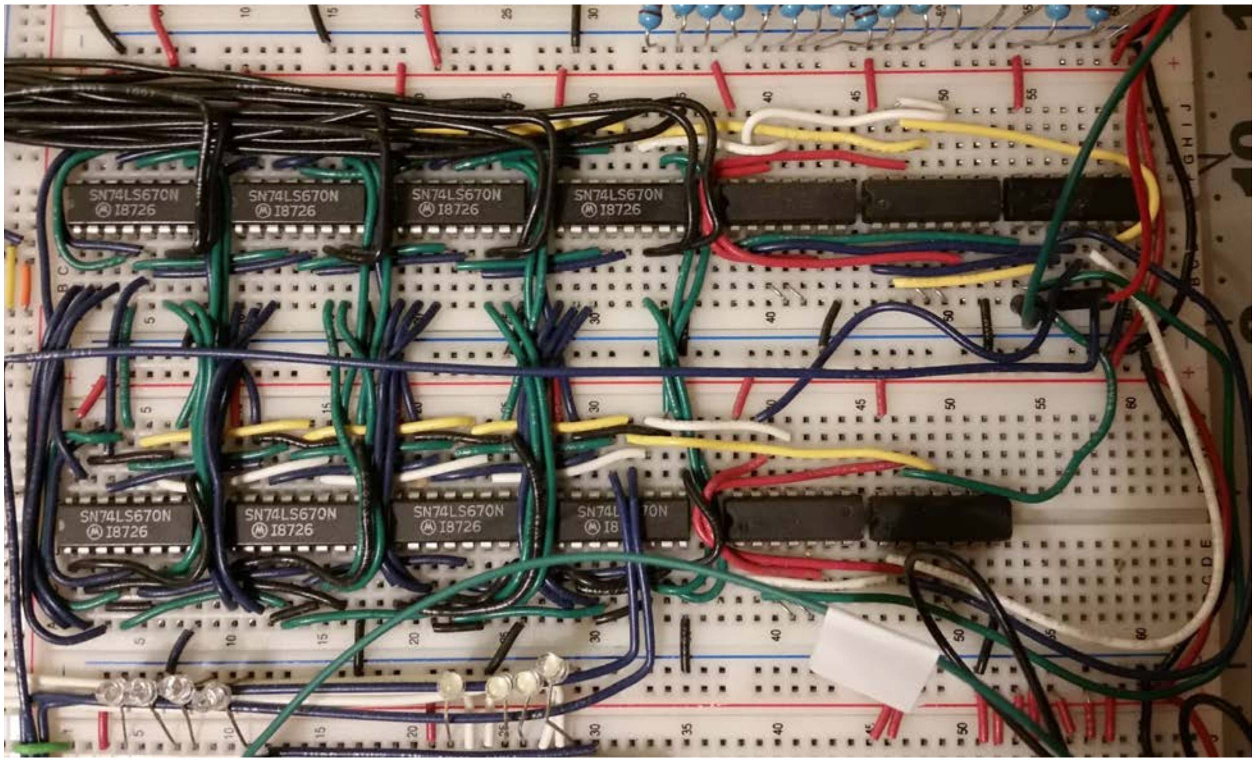


Figure 15. Register file, top-down.

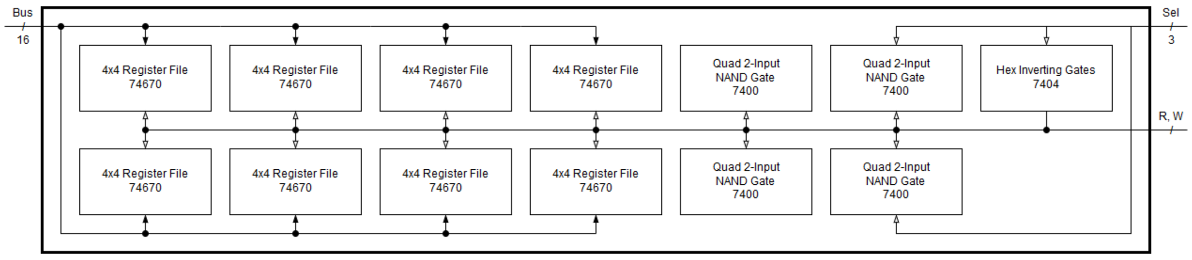


Figure 16. Register file layout.

This circuit contains two sets of four 74LS670, where each set stores four 16-bit registers. Their inputs and outputs are tied together to bus. A tri-state buffer is not used, so there is no way to see into a selected register without sending its' data to the bus. Each set of the four 74LS670 has four control lines: read, write, and two bits to select a register. The 74LS670 allows for simultaneous read and write, but it is not used since reading and writing will not occur at the same time. To avoid using a total of eight control lines, their logic was reduced to Equations 1 through 4, shown below

$$r_0 = r' + w + a_2 \tag{1}$$

$$w_0 = w' + r + a_2 \tag{2}$$

$$r_1 = a_2' + r' + w \tag{3}$$

$$w_1 = r2' + w' + r \tag{4}$$

where a_2 represents the most significant bit of which register to select, *r* is the read enable, and *w* is the write enable. These reduced equations were implemented with four 74LS00 NAND gates and one 74LS04 hex inverter.

4.7 Arithmetic Logic Unit

The arithmetic logic unit (ALU) allows for 16 mathematical and logical operations, has one source operand (SR) register, and one output register. The ALU takes its' second operand from the bus. It is built from six octal d-type latch 74LS373 and four 4-Bit ALU 74LS181. Their configuration is shown in Figure 17 and Figure 18.

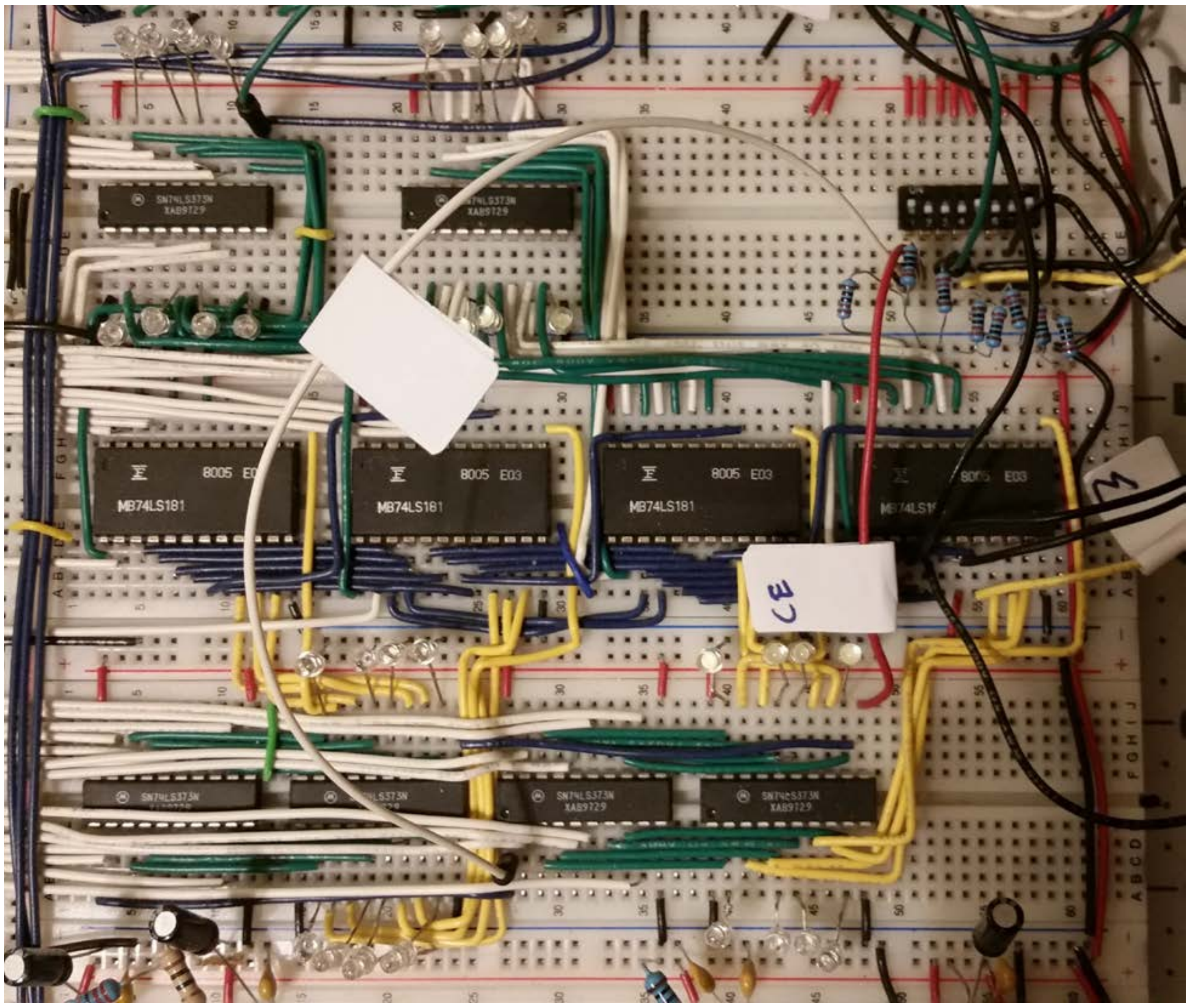


Figure 17. ALU, top-down.

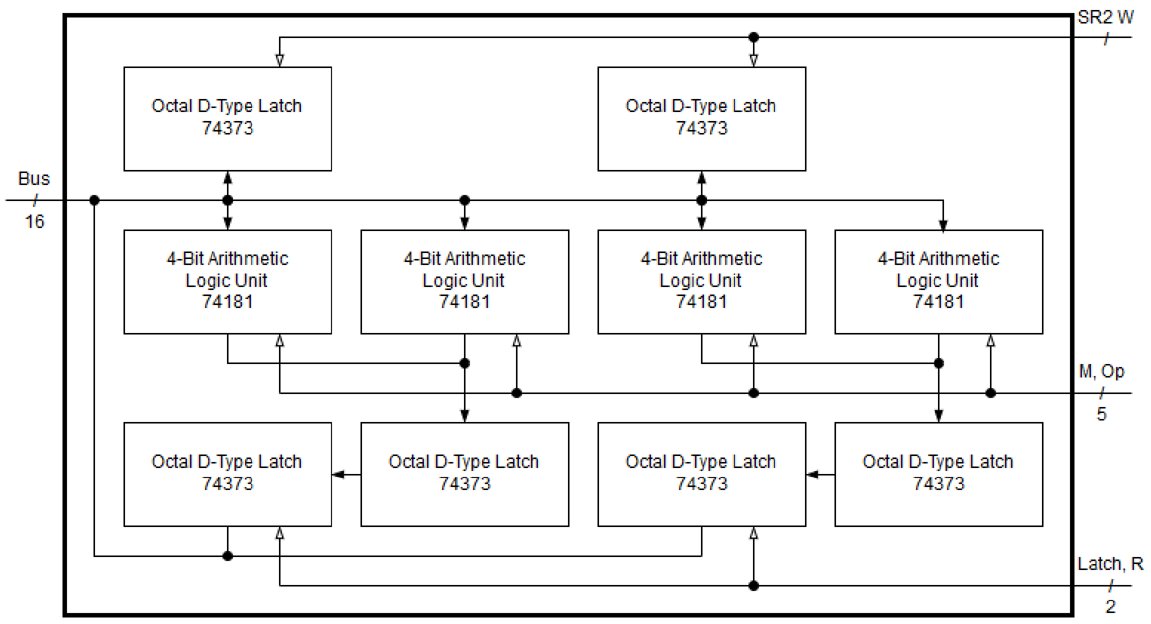


Figure 18. ALU layout.

Four 4-bit arithmetic logic unit 74LS181s were chosen to perform data manipulation operations. Alternatively, mathematical operations could have been implemented through an array of full adders and the logical operations could be derived from available logic gate ICs. However, this would have increased cost and required additional time for implementation and troubleshooting. Therefore, it was decided to move forward with the 74LS181 to expedite development of future modules.

The 74LS181 provides sixteen operations, with modes for active high or active low operands. The pins used include the four input and output lines, mode select, four function select lines, carry in, and out lines. Future features include support for XOR, AND, ADD, NOT, and L/R Shift with active high operands only. This will allow the control line count for mode selection to be reduced from five to three. Two signals, carry generate and carry propagate were not used. These signals implement a lookahead carry generator 74LS182 for quicker calculations. This capability will be considered for future features.

Six octal d-type latch 74LS373s were used instead of the previously used 74LS173. These were chosen because they do not need to read and write simultaneously and have a lower signal overhead. In this module, two octal latches stored the second source operant for the ALU. The remaining four are split into pairs; two contain the low byte of the value computer and two contain the high byte. This allows the computed value to be displayed through LEDs without sending the data to the bus.

5.0 CONCLUSION

Building the 16-bit computer was a challenging project that improved my research, planning, and organizational skills. This included choosing the most efficient parts and schemes, determining what order to build the modules, and staying organized as modules were added to the computer. The computer is not perfect and more efficient implementations exist. In the beginning, progress was slow, but newer modules were organized and worked on the first try. Making progress on a multi-month project that used information from previous classes was satisfying. I am eager to utilize this experience, not just to complete the computer, but also in my future career.

REFERENCES

[1] Y. N. Patt and S. J. Patel, Introduction to Computing Systems, New York: Elizabeth A. Jones, 2004.

[2] J. L. Hennessy and D. A. Patterson, Computer Architecture A Quantitative Approach, Waltham: Elsevier, 2012.