System Hardware

Final Report

4-bit Computer

Lucas Perin (40297625)

Jack Di Spirito (40287812)

Department of Software Engineering and Computer Science

Presented to Achal Patel on June 28th, 2024

Abstract

This report documents the construction and functional evaluation of a 4-bit computer system developed for the SOEN228 lab component. The project consists of conceiving a functional computing system using discrete digital components and a microcontroller, focusing on integrating essential modules such as data registers, a memory address register (MAR), program memory, a control signal generator and much more. Each component's functional contribution to the overall system is explored, emphasizing practical implementation over internal workings. The report serves as a technical manual, providing detailed schematics, timing diagrams, a program listing, and a block diagram essential for replicating the computer. Designed for readers with technical proficiency in circuit wiring, the document ensures clarity and precision in conveying assembly instructions and operational insights without excess detail.

Table of Content

Introduction

Building a 4-bit computer is a basic exercise in comprehending the core concepts of computing systems in the field of digital electronics and computer architecture. The main goal of the SOEN228 Lab was to assemble and understand the fundamentals of computer architecture, and this report is the result of those efforts. Hardware components including a program counter, data registers, memory, and a control signal generator had to be carefully integrated for the project. Every element was meticulously connected on a breadboard to replicate the fundamental operations of a computer system that could carry out a predetermined set of instructions by means of systematic construction and practical experimentation.

This text functions as a thorough technical handbook, meant to offer organized and unambiguous instructions for duplicating and comprehending the constructed 4-bit computer. This report offers comprehensive program listings, timing diagrams, and thorough schematics, with an emphasis on real-world application rather than theoretical complexities. Its goal is to help readers who have a rudimentary understanding of circuit wiring build a comparable computer system and understand its workings. The parts of the computer that follow will be discussed in detail, along with their functions, connections and methods of functioning.

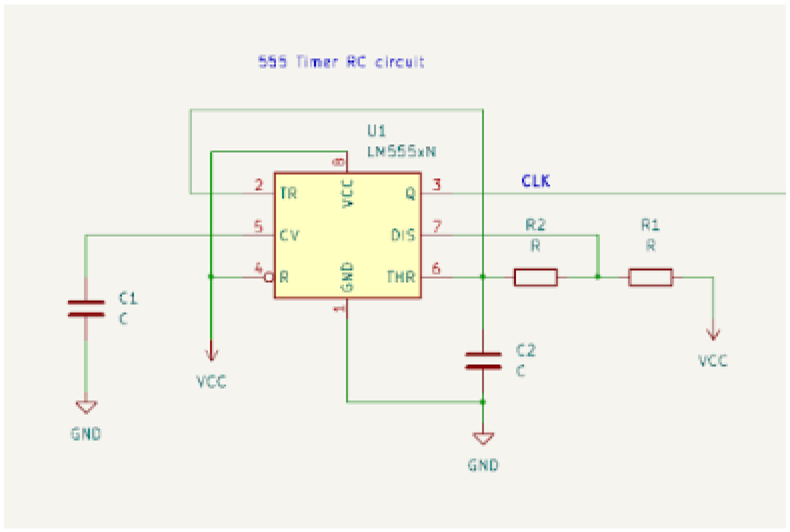
Timing Signal Generator

The first essential component of the 4-bit computer that needed to be built was the Timing Signal Generator. Sort of like the heartbeat of the computer, this component's main job is to generate a sequence of accurate timing signals that coordinate the actions of different computer components. To build it, three main components need to be set up on the breadboard: the 555 timer in astable mode, the 74LS164 SIPO (Serial-In-Parallel-Out) shift register, and the 7420 quad input NAND gate. Each of these play a crucial role in generating the electrical pulses that will control our computer components.

555 Timer in Astable Mode

When set up in astable mode, the 555 timer produces a continuous square wave output that is used as the circuit's clock signal by acting as an oscillator. The resistors R1 (22 kΩ) and R2 (56 kΩ), along with the capacitor C2 (3.3 μF) in the RC network linked to the 555 timer, determine the frequency of this clock signal. Furthermore, the circuit's noise is reduced by capacitor C1 (10 nF). The output signal's high (logic 1) and low (logic 0) timing intervals are determined by the cycles of charging and discharging capacitor C2. *Figure 1* illustrates how these electrical components should be connected.

Fig.1 Astable 555 timer circuit



The frequency of the clock signal can be calculated using the formula:

$$f_s = \frac{1}{T_s} = \frac{1.44}{(R_1 + 2R_2)C_2}$$

= 1.44/(22000Ω+112000Ω)3.3×10−6F = 3.26Hz

Similarly, the duty cycle (DC) of the output signal is determined by:

$$DC = \frac{t}{T_s} = \frac{R_1 + R_2}{R_1 + 2R_2}$$

= 78000Ω/134000Ω ≈ 0.582 ≈58.2%

74LS164 SIPO Shift Register

The 74LS164 SIPO shift register receives the output clock pulse from the 555 timer as its clock input. Data is serially shifted from the input by this shift register and shown on eight parallel outputs (QA to QH) as shown in *figure 3*. The data at the serial input is moved into the register's first flip-flop on each rising edge of the clock pulse, while the data in each flip-flop is shifted to the subsequent position in the sequence. As a result, every clock cycle, the data propagates through the register.

7420 NAND Gate with Quad Input and Feedback Mechanism

A 7420 quad input NAND gate is used as part of a feedback mechanism to create a certain order for the shift register outputs. The NAND gate's inputs are coupled to the shift register's outputs QB, QD, QF, and QG. These inputs are processed by the NAND gate, which then outputs a result that is fed back into the shift register's serial input.

Because of its logic characteristics, the NAND gate initially generates a high output when all of the outputs are zero (a NAND gate outputs high unless all of the inputs are high). The shift register fills with high states as the clock pulses continue. The output of the NAND gate changes to low when QB, QD, QF, and QG all become high. The serial input of the shift register is then fed back with this low output.

This feedback loop makes sure that a low state is added after the register is filled with high states, causing a pulse of low states to move through the register. The operation of the circuit is shown visually by the LEDs attached to the shift register's parallel outputs (QA to QH). In other words, the pattern of a group of “two 0s moving through a sea of 1s” should be outputted on an LED. At first, all of the LEDs are off, reflecting the state of the shift register. The matching LEDs illuminate in tandem with the change of the high states through the register. A moving pattern of two off LEDs inside a sequence of on LEDs is produced when the feedback introduces a low condition, graphically illustrating the timing signals being generated, which can also be seen in the timing diagram in *figure 3*.

Fig.2 Schematic for Time Signal Generator with 555 Timer, 74LS164 SIPO Shift Register and 7420 NAND Gate

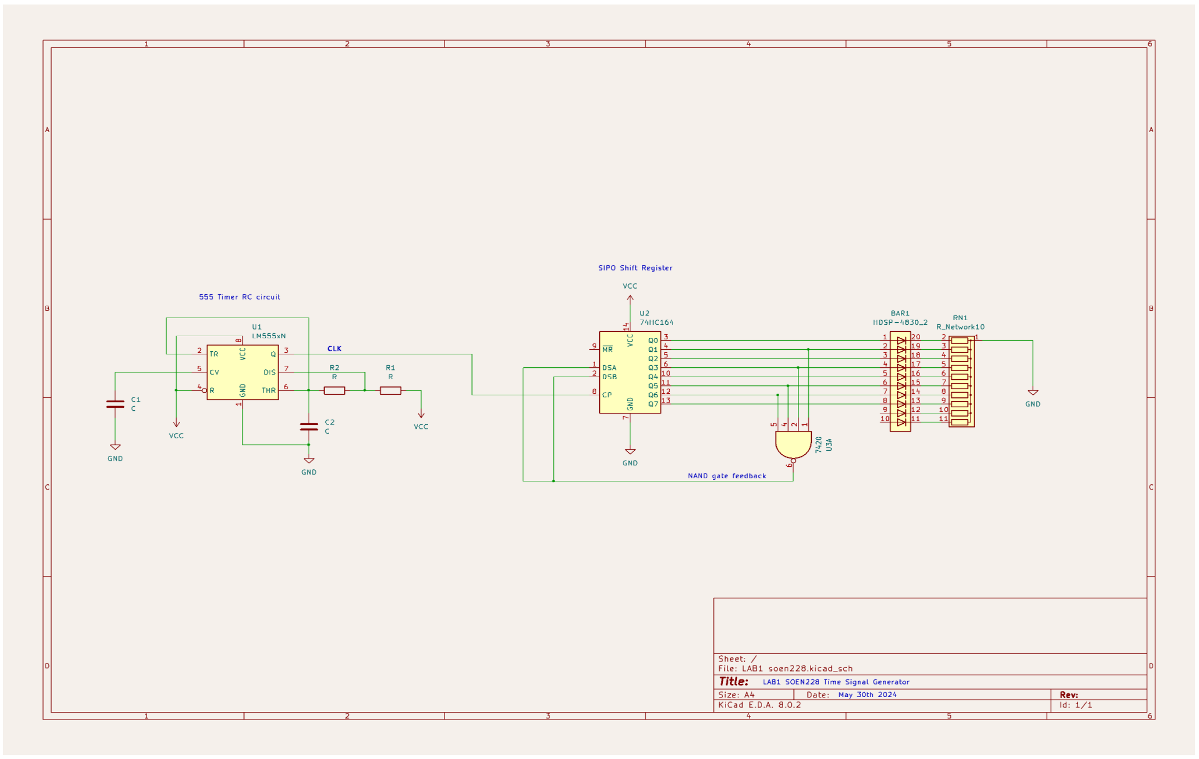


Fig.3 Timing diagram for fig.2

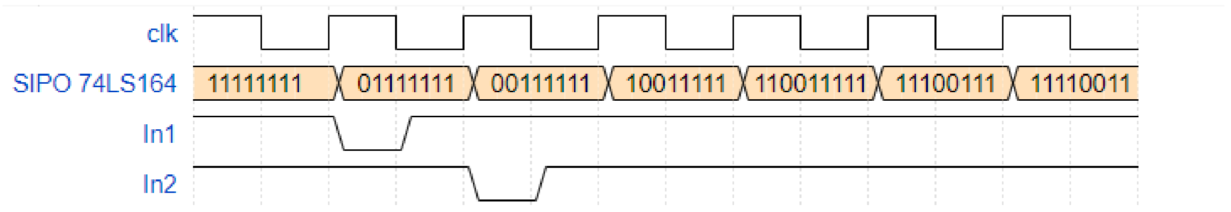


Fig.4 Pins for 555 Timer IC

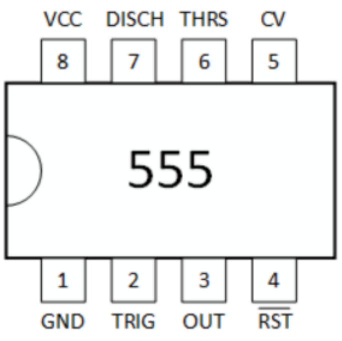


Fig.5 Pins for 74LS164 SIPO Shift Register

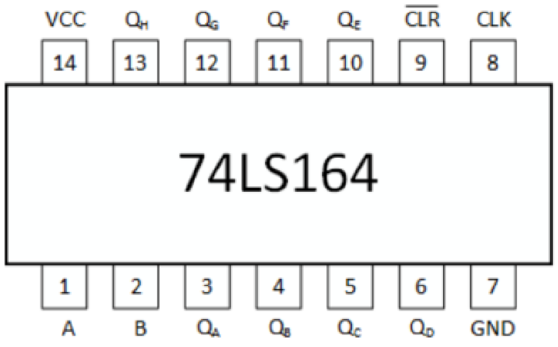
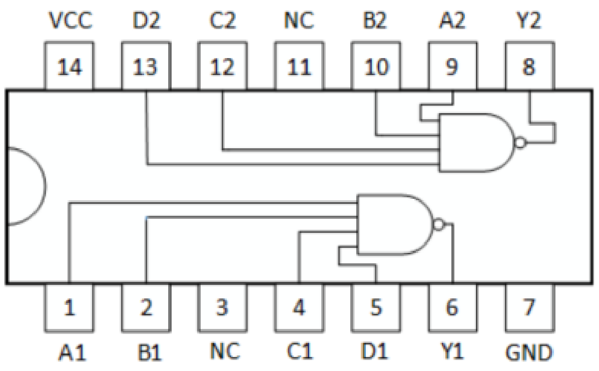


Fig.6 Pins for 7420 4-1 NAND



Data Bus

In our 4-bit computer, the data bus is an essential communication channel that joins different parts and enables effective data flow between them. Tri-state logic is used to create the bus, which is intended to handle 4-bit data transfers, in order to prevent bus conflicts and guarantee trouble-free operation. To make the bus, 4 wires will be used (one for each bit of data) and are to be connected along the width of the breadboard *(see figure 7)*.

Tri-State Logic and Bus Operation

Using tri-state logic is crucial for controlling several devices on a single bus. The intended ICs already have the tri-state built into them. Only when its control signal is active (high) may any device connected to the bus drive its output onto the bus. The device's output is in a high-impedance condition while the control signal is low, which essentially disconnects it from the bus to avoid interfering with other devices.

Program Counter and ALU

The arithmetic logic unit (ALU) and program counter are essential parts of our 4-bit computer. Together, they oversee the instructions' sequential execution and carry out necessary mathematical operations to guarantee the computer runs properly.

The Program Counter (PC)

The address of the subsequent instruction to be executed is stored in the program counter, which is implemented using a 74LS173 register. This register is crucial for guiding the computer through the sequence of instructions stored in memory. The timing signals produced by the Timing Signal Generator, namely pulses T0 through T3, are synchronized with the program counter's operation visualized in the timing diagram in *figure 8*. Essentially, the PC is simply a register that points to the next instruction that is to be executed. Every instruction cycle, it is incremented, thus demonstrating the need for an ALU, which is the combination of registers that actually increment the contents of the program counter. This is done in 4 steps, called the fetch phase:

T0 (PC out): The program counter register places its current value onto the bus

T1 (Sum In): The sum register captures the value that's currently on the bus and increments it PC+1 using the arithmetic unit.

T2 (Sum out): The sum register outputs its current value onto the bus (post-increment).

T3 (PC In): The program counter takes the value currently on the bus and stores it as its new value.

T4 … T7 : These timing signals stay idle until further notice.

The Arithmetic Logic Unit (ALU)

The 4-bit computer's ALU is made up of a SUM register (74LS173) and a 74LS283 4-bit adder. Additionally, another 74LS173 register called the Mirror Register will be used in order to display the actual incrementing instructions from 0-15 on an LED to ensure proper functioning of this cycle. During the fetch phase, the ALU’s main job is to move the program counter forward. It functions in tandem with the program counter as follows:

T1 (Sum In): The ALU receives the current address value from the data bus and increments it by one.

T2 (Sum out): The ALU places the incremented address back onto the data bus.

To assemble these 4 registers together, the schematic in *figure 7* demonstrates all the critical connections to be made pin-wise and exactly where each timing signal (T0-T3) should be inputted for now.

Fig.7 Schematic of Bus, ALU and PC

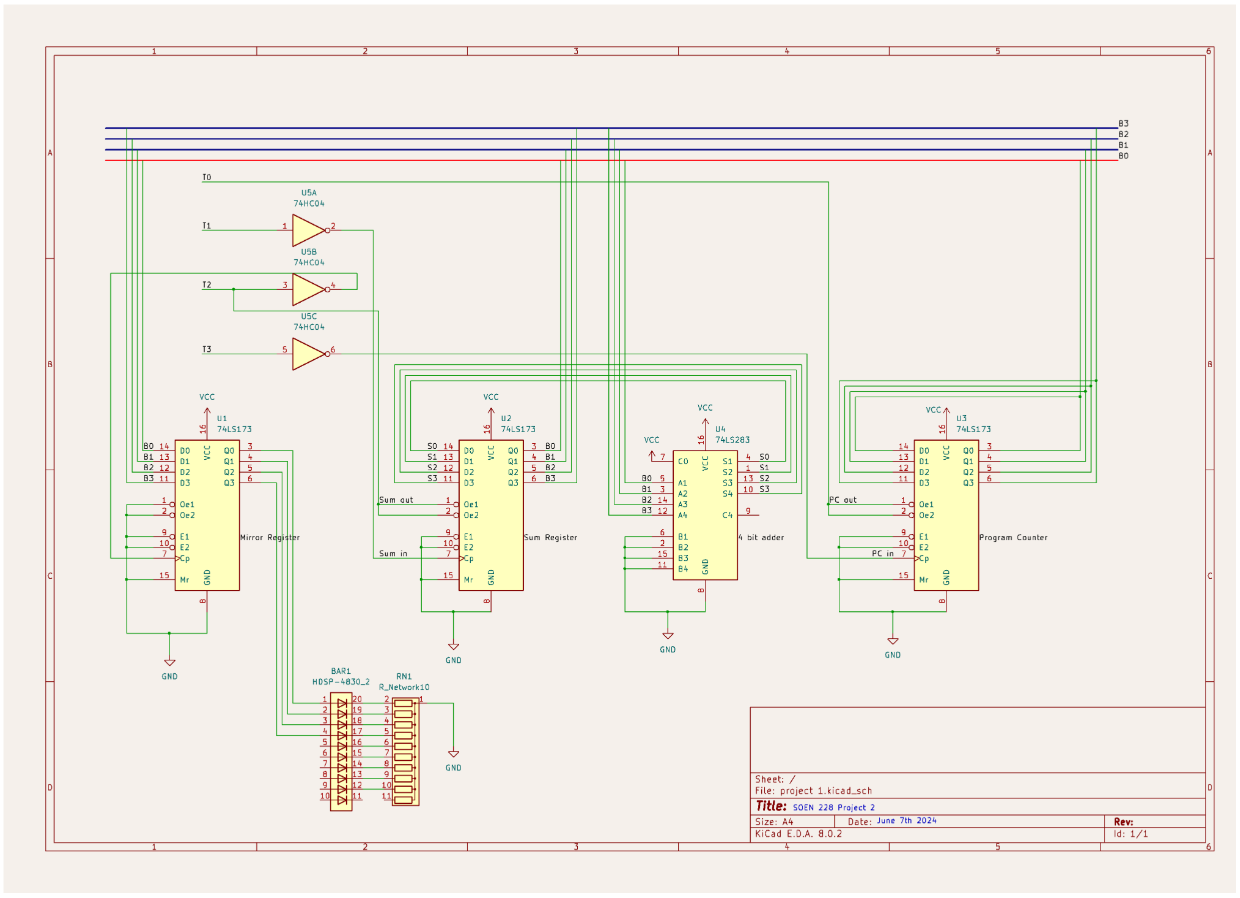


Fig.8 Timing diagram showing one instruction cycle for Clk, Bus, SR, PC and T0…T7

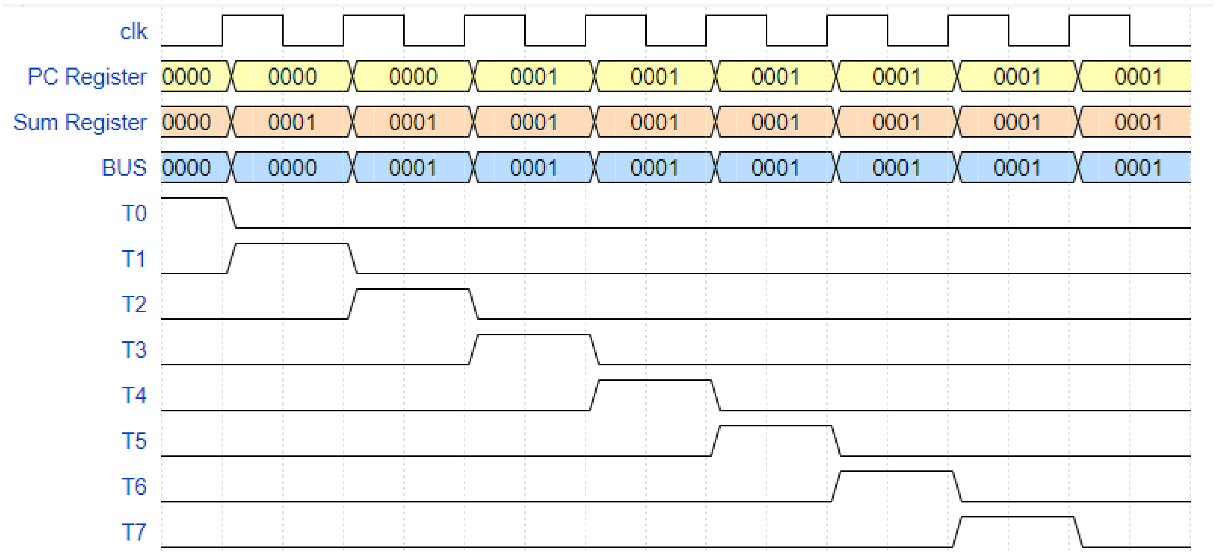

clk
PC Register
Sum Register
BUS
T0
T1
T2
T3
T4
T5
T6
T7
PC Register: 0000 0000 0000 0001 0001 0001 0001 0001 0001
Sum Register: 0000 0001 0001 0001 0001 0001 0001 0001 0001
BUS: 0000 0000 0001 0001 0001 0001 0001 0001 0001

Fig.9 Pin-out diagram for 74LS283 4-bit adder

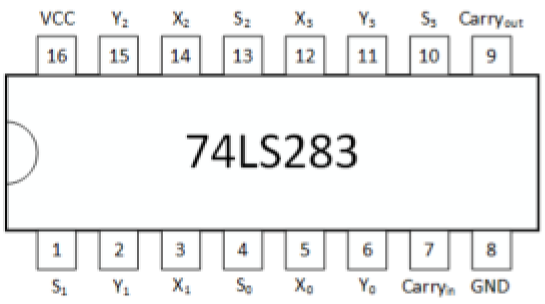
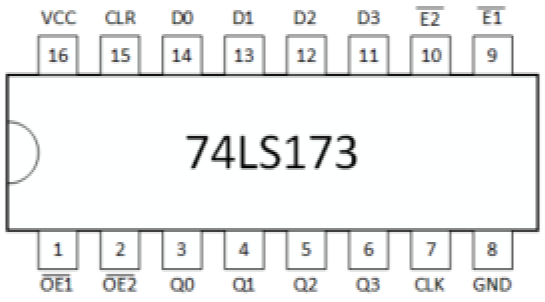


Fig.10 Pin-out diagram for 74LS283 Register



Onboard Registers

The incorporation of onboard registers is essential to controlling memory operations and data flow in the 4-bit computer project. Two 74LS173 4-bit registers are to be implemented in this step - the process: the Memory Address Register (MAR), and two Memory Data Registers (MDR) called A and B.

Memory Address Register (MAR)

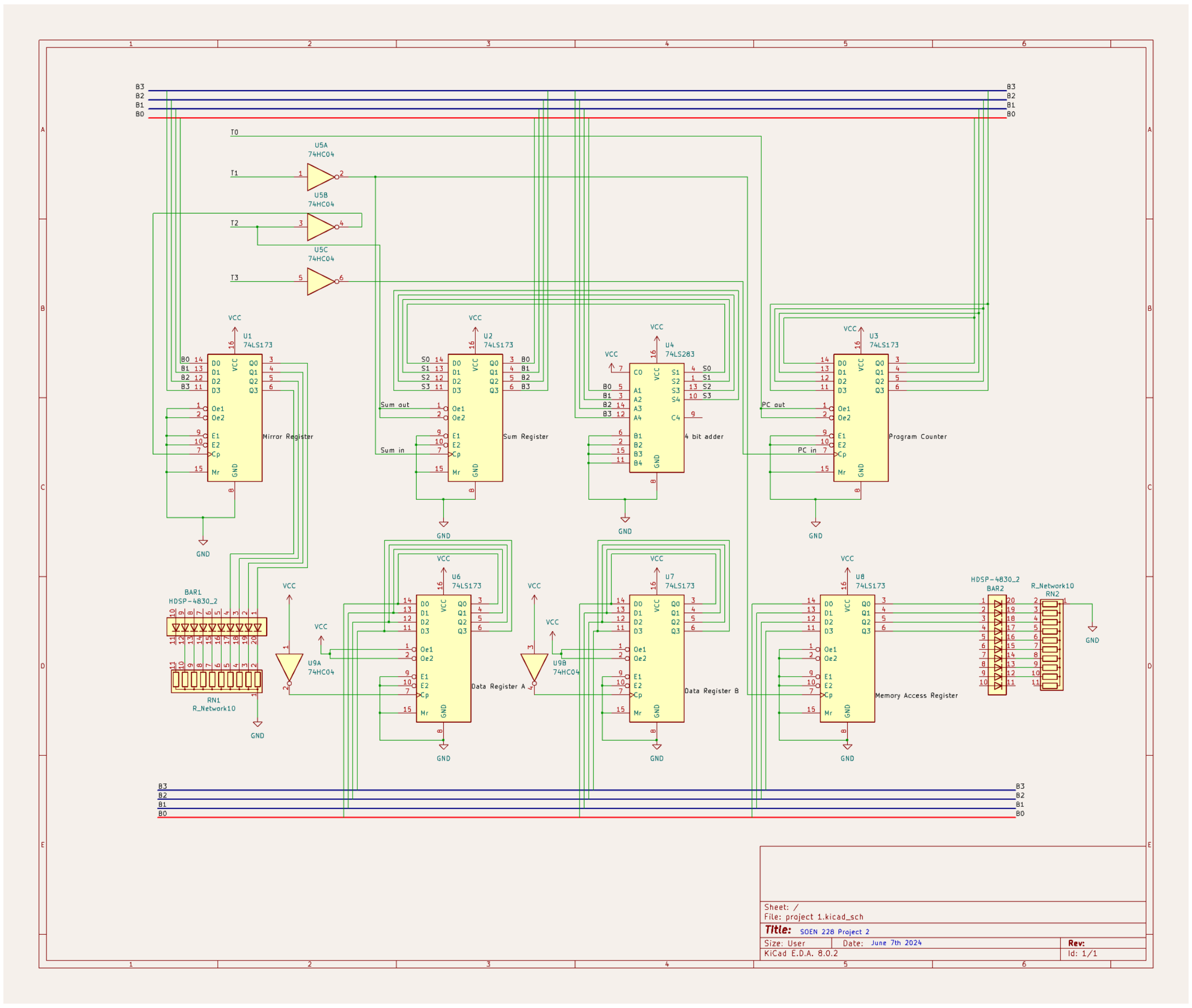
The Memory Address Register (MAR), designated as Register A, stores the memory address that the computer is currently using to read instructions or data. It is connected directly to the bus following the pinout diagram in *figure 10* and the schematic in *figure 11*.

The MAR uses pulses from the time Signal Generator to synchronize its operation with the rest of the components. The computer will only access memory addresses precisely and on schedule thanks to this synchronization. It functions similarly to a pointer, pointing the computer in the direction of certain data or instructions kept in memory. In this case, as seen in *figure 11*, the MAR is connected to timing pulse T1. This is because at T1, the current address in memory is on the bus. This is the current instruction that is needed to be stored in the MAR, because at T2 and T3, the instruction will be incremented to point to the next instruction.

Memory Data Register (MDR)

Complementing MAR, the Memory Data Registers (MDR), also known as Register A and B, store data that is transferred to or retrieved from memory in a temporary manner. In essence, whatever data is stored at the address inside the MAR gets stored in the MDR. Similar to MAR, MDR is bus-connected as well. The data registers A and B will ultimately be victim to the instructions given by the CPU, however, these instructions will be covered in detail later in the document.

Fig.11 Schematic of Data Registers A and B and MAR added to fig.1 configuration



Memory

In the 4-bit computer project, the ATTiny2313A microcontroller is configured as a 16 x 4-bit ROM (Read Only Memory) as the focal point of the memory subsystem. The microcontroller comes preloaded with some firmware (16 instructions) and will act as the CPU of the 4-bit computer. It outputs corresponding 4-bit data on pins PB0 to PB3 *(see fig 17)* and receives 4-bit addresses (0–15) from the Memory Address Register (MAR). The 4-to-16 line decoder receives this data, which contains machine instructions, and uses the 4-bit input to activate one of its 16 output lines. However, for this project, the 7442 decoder only produces 10 outputs from 4 inputs. For instance, the ATTiny2313A will output 0101 in response to an input address of 0101 (0x5), which will activate line 5 of the decoder. Since the 7442 is active low, the decoder translates binary inputs into one-hot encoded outputs, which are binary representations where only one bit is *'0'* (low) at a time, crucial for selecting specific operations in sequential logic circuits.

ATTiny2313A microcontroller

The outputs of the MAR should be connected to the inputs of the microcontroller so that it can access the ROM to retrieve the 4-bit data stored at that location inputted by the MAR *(see schematic figure 14)*. For example, if the PC points to address 0x3 (binary 0011), the microcontroller fetches the binary data 0011 from that memory location. This data indicates a specific data value or a machine instruction needed for the program to process it further. Here is the mapping of each memory location within the microcontroller with respect to the inputs PB0-PB3 coming from the MAR:

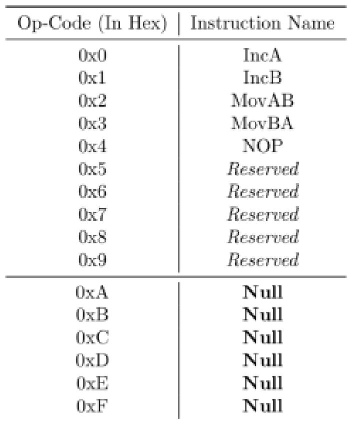
Fig.12 Sample Layout for 16 X 4 bit Memory (our case)

Address	**PB0**	**PB1**	**PB2**	**PB3**
0x0	*0*	*0*	*0*	*0*
0x1	*0*	*0*	*0*	*1*
0x2	*0*	*0*	*1*	*0*
0x3	*0*	*0*	*1*	*1*
0x4	*0*	*1*	*0*	*0*
…	...	...	...	...
0xF	*1*	*1*	*1*	*1*

7442 Decoder

The outputs of the ATiny microcontroller are then inputted into the 4-10 decoder such that it can output only one of 10 data lines *(see fig 15)* which each correspond to an instruction or opcode. For example, if line 3 (0x3) is activated by the decoder, the instruction corresponds to a MOV B, A instruction or if line. This is illustrated in the following table:

Fig.13 Instruction Set



Op-Code (In Hex)	Instruction Name
0x0	IncA
0x1	IncB
0x2	MovAB
0x3	MovBA
0x4	NOP
0x5	*Reserved*
0x6	*Reserved*
0x7	*Reserved*
0x8	*Reserved*
0x9	*Reserved*
0xA	**Null**
0xB	**Null**
0xC	**Null**
0xD	**Null**
0xE	**Null**
0xF	**Null**

Fig.14 Schematic of Program Memory Components

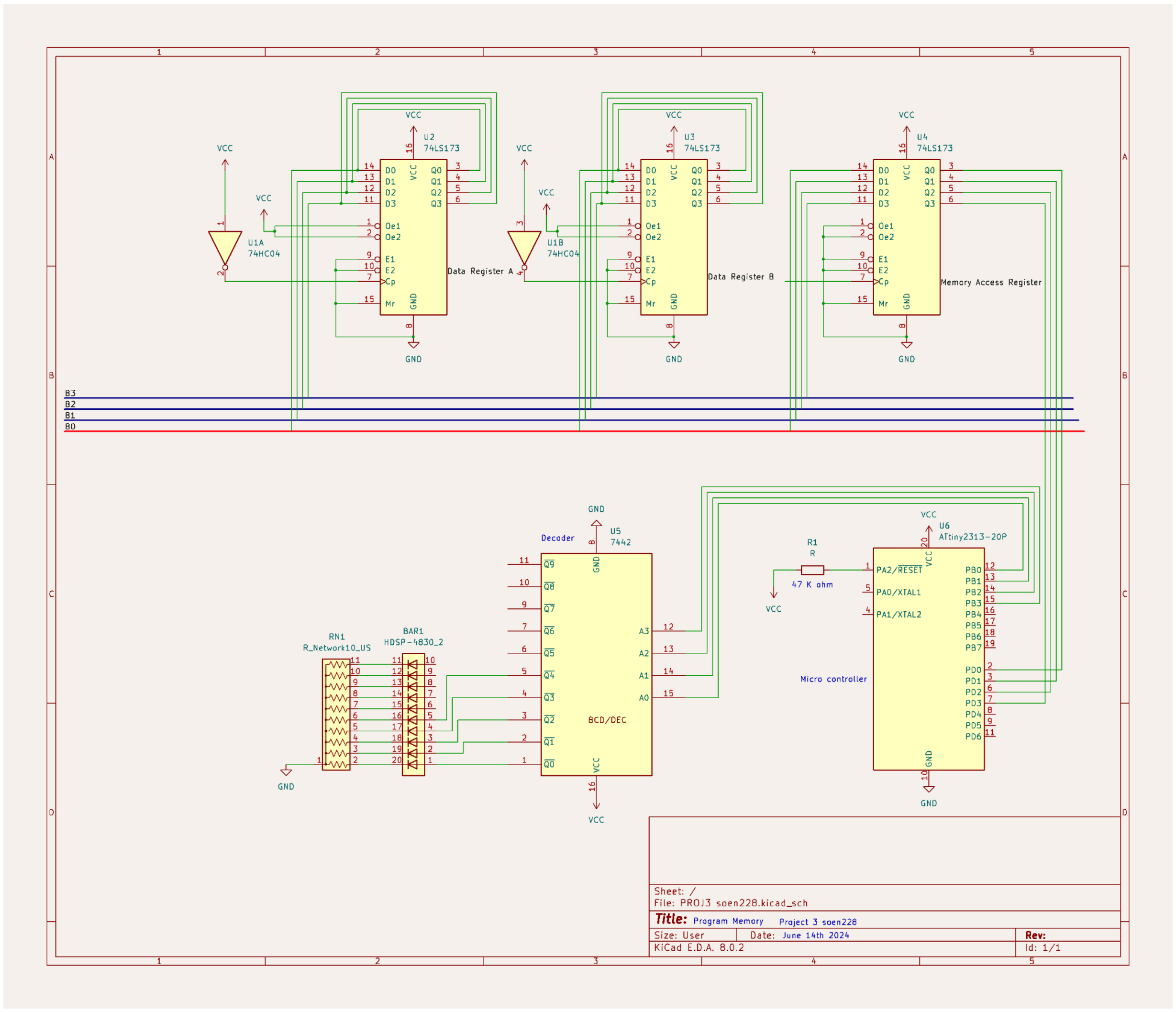


Fig.15 Truth Table for 7442 Decoder

I_3	I_2	I_1	I_0	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$	$\overline{Y_4}$	$\overline{Y_5}$	$\overline{Y_6}$	$\overline{Y_7}$	$\overline{Y_8}$	$\overline{Y_9}$
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig.16 Pin-out diagram for 7442 decoder

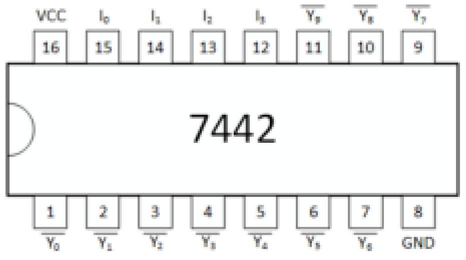
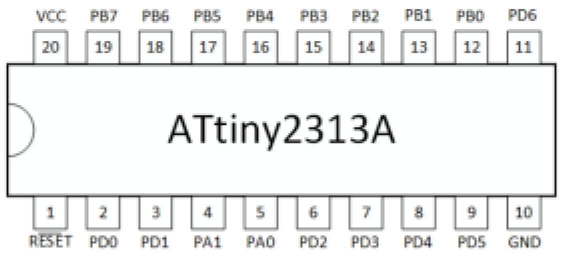


Fig.17 Pin-out diagram for ATiny2313A



Once the two ICs are properly wired, the outputs of the decoder can be connected to another LED in order to visualize the sequential instructions being executed by the program that is preloaded in the microcontroller.

Control Signal Generator

Until now, the opcodes are fetched from the memory of the microcontroller and displayed onto an LED after being decoded. The control signal generator will add functionality to these opcodes being sequentially read by the microcontroller. For this, timing signals T4 to T7, which were previously idle, will be made useful. Following *figures 26 and 27* will be useful for the proper connections. In addition to this, three 7432 “OR” gate ICs, two 7408 “AND” gate ICs and a 7404 Inverter are to be implemented to make all the combinational logic required to generate control signals that enable the execution of instructions decoded from memory. *(see figures 18,19 and 20).*

Fig.18 Pins for 7404 inverter

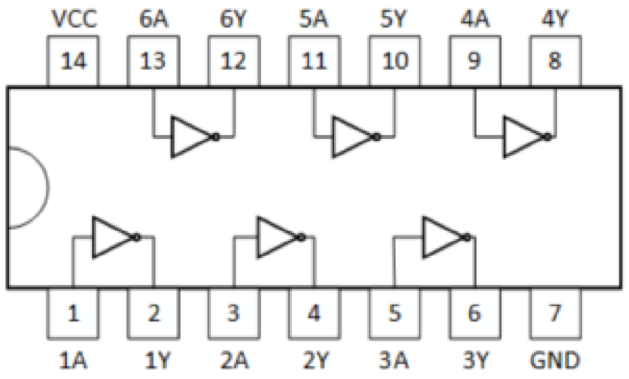


Fig.19 Pins for 7408 AND

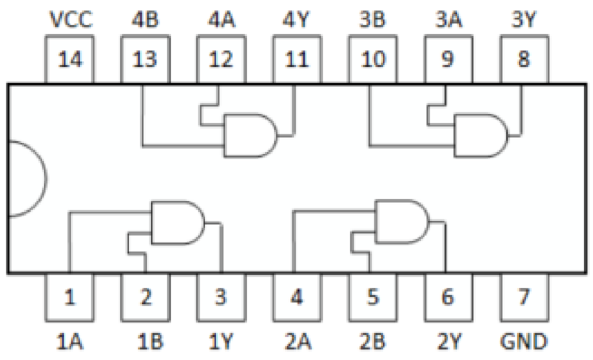
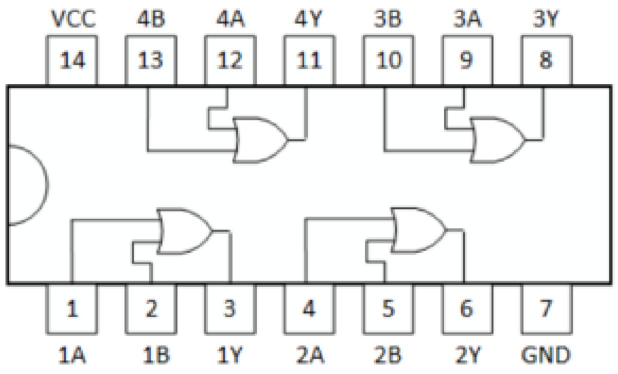


Fig.20 Pins for 7432 OR

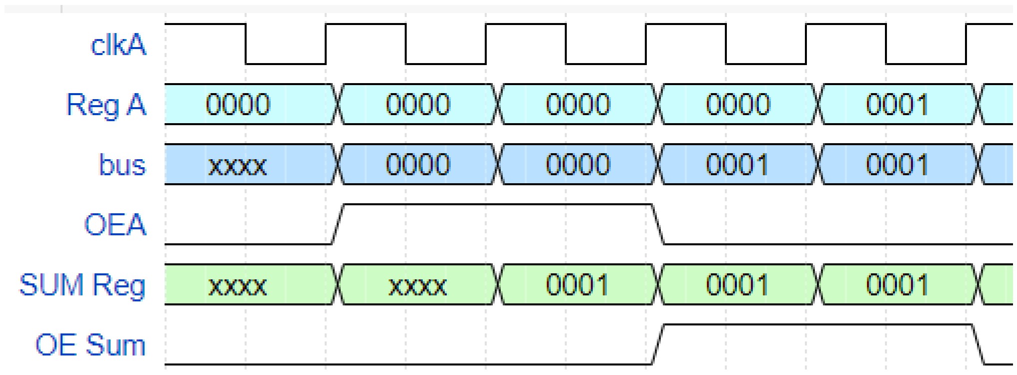


Overview of Instructions

Inc A

1. Data Register A outputs its contents onto the bus.
2. The SUM Register stores the incremented value of A (A + 1).
3. The SUM Register outputs the incremented value (A + 1) onto the bus.
4. Data Register A stores the incremented value from the bus.

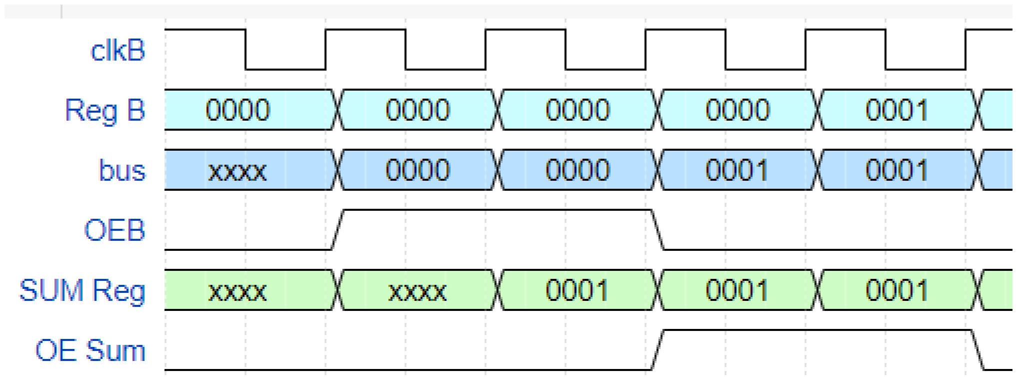
Fig 21 timing diagram for INC A



Inc B

1. Data Register B outputs its contents onto the bus.
2. The SUM Register stores the incremented value of B (B + 1).
3. The SUM Register outputs the incremented value (B + 1) onto the bus.
4. Data Register B stores the incremented value from the bus.

Fig 22 Timing diagram for INC B



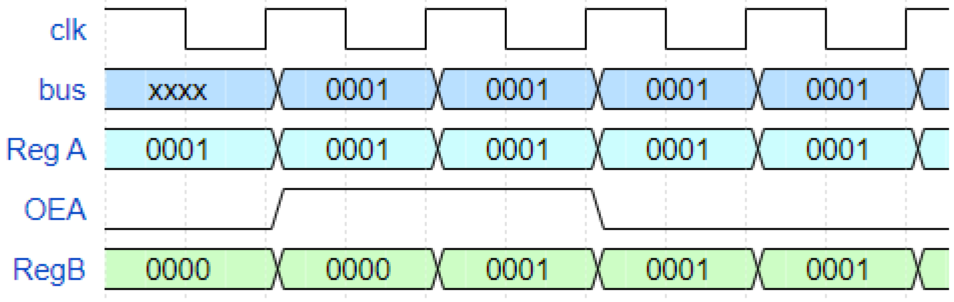
Mov A, B

The MovAB (Move A to B) instruction copies the value of Data Register A to Data Register B.

This operation is achieved through the following steps:

1. Data Register A outputs its contents onto the bus.
2. Data Register B stores the contents from the bus.

Fig 23 timing diagram for MOV A,B



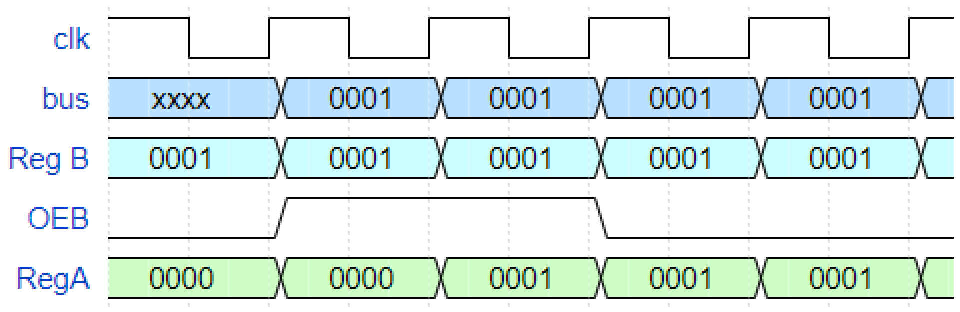
Mov B, A

The MovBA (Move B to A) instruction copies the value of Data Register B to Data Register A.

The execution steps are:

1. Data Register B outputs its contents onto the bus.
2. Data Register A stores the contents from the bus.

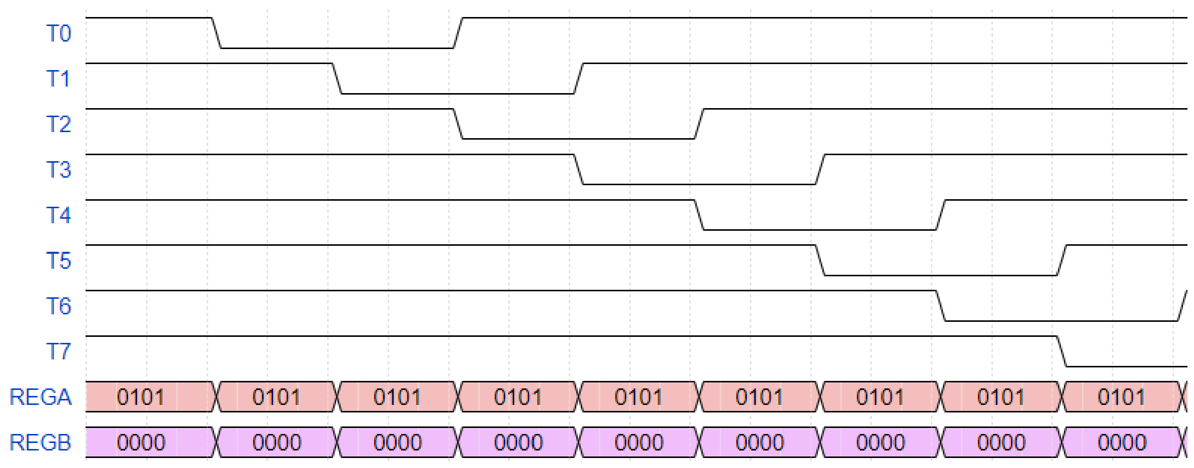
Fig 24 timing diagram for MOV B,A



NOP

The NOP (No Operation) instruction simply acts as a delay in the execution sequence.

Fig 25 timing diagram for NOP



As mentioned above, the combinational logic that governs the physical operation of these instructions is given by the following boolean functions:

Data Register A
$OE_A = (T_4 + I_{sig0}) \cdot (T_4 + I_{sig2})$
$CLK_A = (T_7 + I_{sig0}) \cdot (T_5 + I_{sig3})$
Data Register B
$OE_B = (T_4 + I_{sig1}) \cdot (T_4 + I_{sig3})$
$CLK_B = (T_7 + I_{sig1}) \cdot (T_5 + I_{sig2})$
Sum Register
$OE_{SUM} = (T_6 + I_{sig0}) \cdot (T_6 + I_{sig1}) \cdot T_2$
$CLK_{SUM} = (T_5 + I_{sig0}) \cdot (T_5 + I_{sig1}) \cdot T_1$

Each register has two control signals. An output enable (OEx), which dictates when the register should output its data onto the bus, and a specific clock signal (CLKx), which determines when the register in question should store the data being inputted from the bus.

Fig 26 Registers A (left) and B(right) Logic

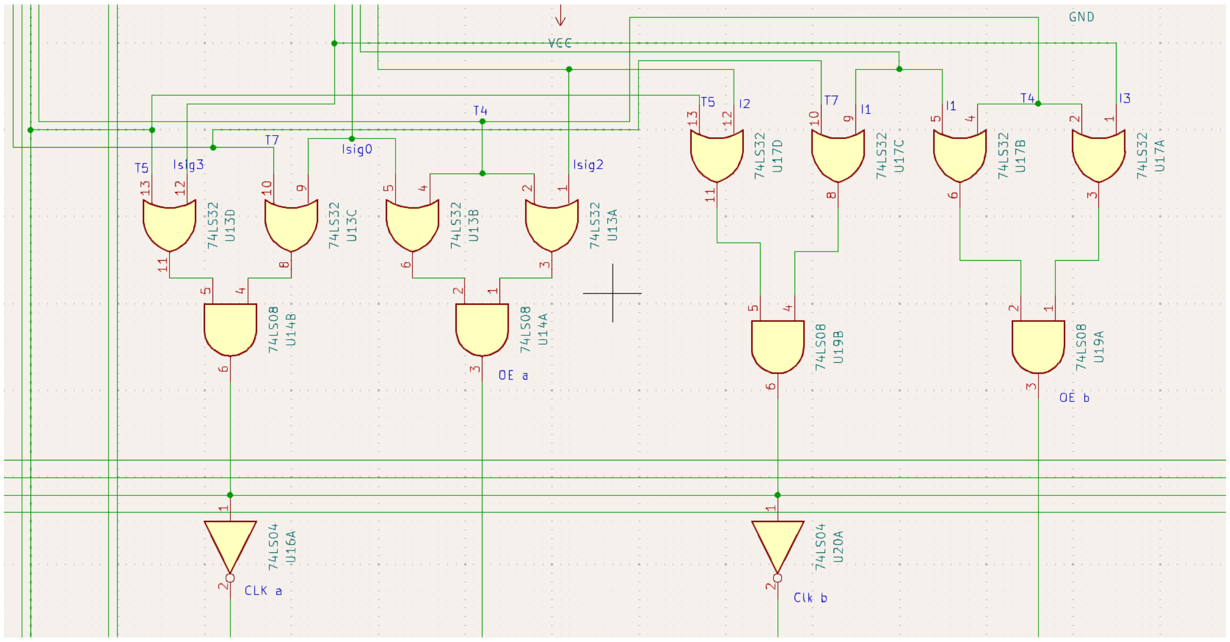
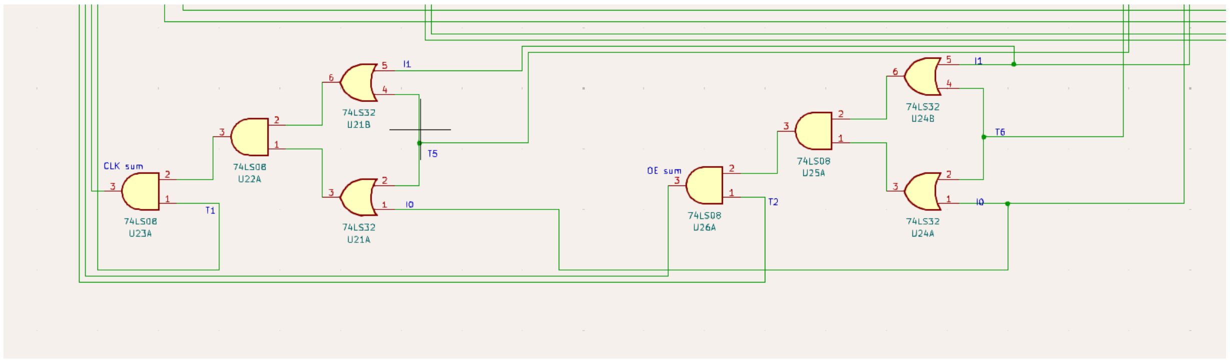


Fig 27 Sum Register logic



Program

After decoding the Assembly Language Program with respect to the binary instructions shown on the LED and the provided instruction set *(See fig. 13)*, the preloaded code in the microcontroller can be deduced. Below is a table showing the contents of the Memory Address Register (MAR), the instruction that is contained in the memory slot pointed to by the MAR, the decoded binary value of that instruction and finally the corresponding assembly language code.

Fig.28 Decoded Program

MAR	Instruction	7442 Decoder	Assembly
0000	0x4	11110	NOP
0001	0x0	01111	INC A
0010	0x0	01111	INC A
0011	0x0	01111	INC A
0100	0x2	11011	MOV A, B
0101	0x1	10111	INC B
0110	0x1	10111	INC B
0111	0x4	11110	NOP
1000	0x3	11101	MOV B, A
1001	0x1	10111	INC B
1010	0x0	01111	INC A
1011	0x1	10111	INC B
1100	0x0	01111	INC A
1101	0x2	11011	MOV A, B

1110	0x3	11101	MOV B, A
1111	0x0	01111	INC A

Overview of the Computer

This 4-bit computer project integrates various components to execute simple instructions, demonstrating fundamental computer architecture concepts. At the core, the 555 timer (see Figures 2 and 4), generates the clock pulses which drive the entire operation of the system created along with the 74LS164 shift register that works in conjunction with the 7420 NAND gate to manage control signals. The first LED pack of the computer highlights specific states of the signals, displaying two zeros amidst ones. This is a crucial aspect of the computer, since it provides proof by visual indication that the state transitions are valid (see Appendix 0).

For the communication and data transfer pathway, a single 4-bit bus is connected to various components, including the program counter, sum register, and mirror register. These registers are implemented using 74LS173 ICs with tristate outputs. The 74LS283 4-bit adder is utilized to perform arithmetic operations. By means of the bus, the program counter outputs its value during specific clock cycles, allowing the sum register to store the increment value. The mirror register, also connected to the bus, holds a copy of the incremented value to display the binary count from 0 to 15. The second LED of the computer displays this increment and ensures the proper functioning of the bus (see Appendix 1).

The memory storage device uses read only memory, which fits well with the sequential nature of the program counter. This register is directly connected to the ATTiny microcontroller whose firmware is critical for memory operations and is not changed for the project. These operations are then sent to the 7442 decoder whose code (see Figure 28), is displayed on another LED on the circuit (see Appendix 3). Last but not least, the Control Signal Generator uses timing signals T0-T7 to ensure precise executions. Using the last AND and OR gates on the board, the proper signals to data registers A and B replace the ground input previously used so that at last, a functional computer remains (see Appendix 4).

Fig.29 4-bit computer schematic

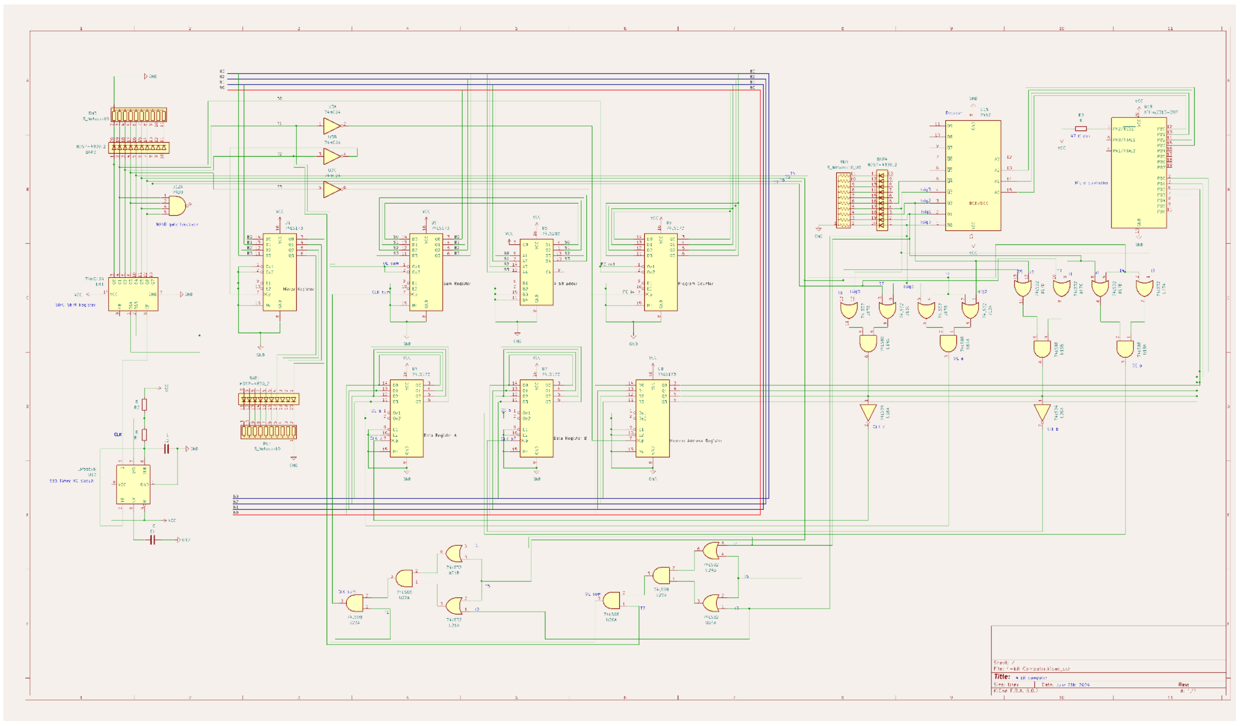
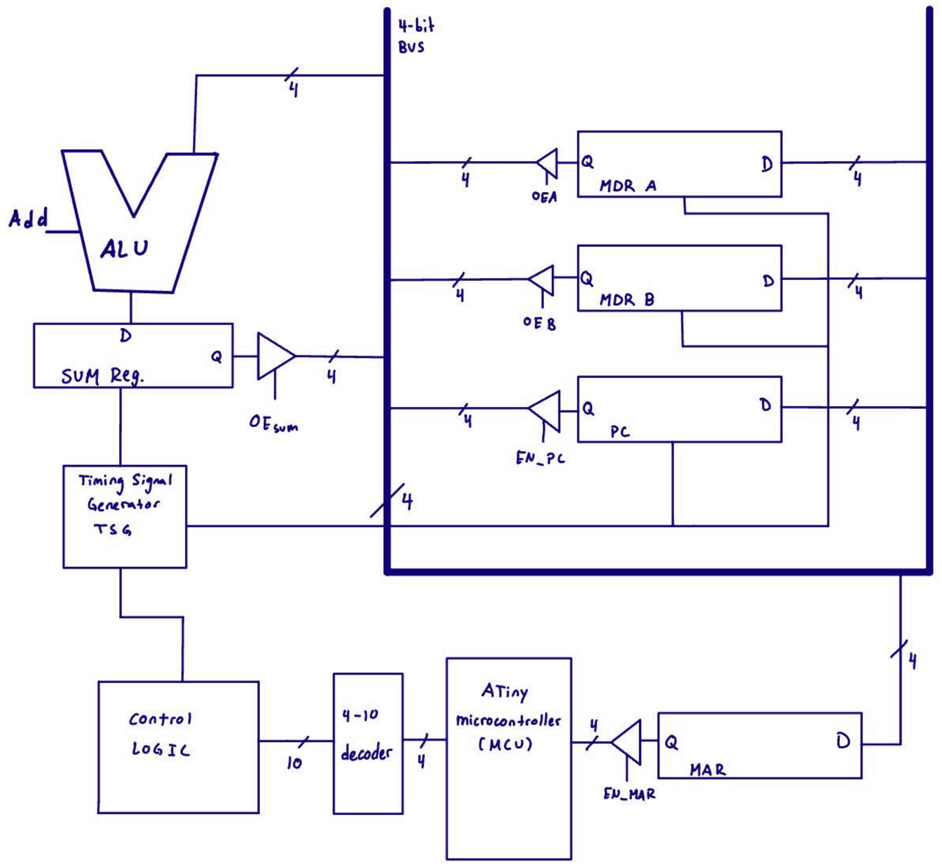


Fig 30 Block Diagram



Operation of the Computer

Fundamentally, the computer operates on a fetch-execute cycle, which involves fetching an instruction from memory, decoding it, and then executing it with respect to the 8 timing pulses generated from the TSG (T0 to T7). Below is the complete fetch-execute cycle for an instruction, describing the detailed operation of the computer.

Fetch

T0 (PC out): The program counter register places its current value onto the bus

T1 (Sum In): The sum register captures the value that's currently on the bus and increments it PC+1 using the arithmetic unit. The current instruction is loaded into the MAR.

T2 (Sum out): The sum register outputs its current value onto the bus (post-increment).

T3 (PC In): The program counter takes the value currently on the bus and stores it as its new value.

Decode

The ATiny MCU receives the address of the current instruction from the MAR as an input and outputs the opcode of that instruction into the inputs of the 4-10 decoder. The decoder then translates that opcode input into 4 instruction signals: Isig 0 to Isig 3. These will be used as control signals to execute whatever instruction that it's representing.

Execute (e.g. Inc A)

T4: The content of data register A is output onto the bus. Output Enable A (OE A) is set high.

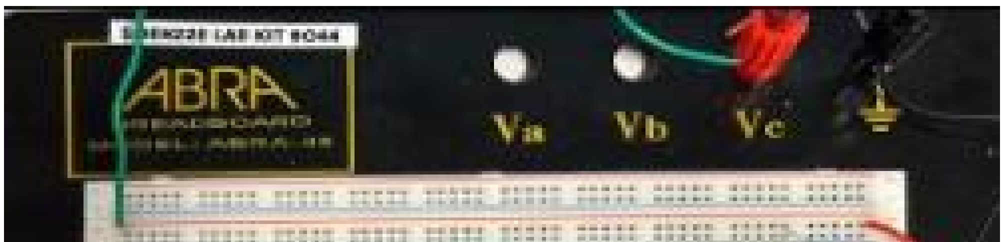
T5: The ALU loads the value on the bus and increments it by adding 1 to it. (S=A+1)

T6: The sum register outputs the incremented value back onto the bus. Output Enable Sum (OE sum) is set high.

T7: The incremented value is loaded back into Register A from the bus with the help of the clock pulse of register A (CLK A).

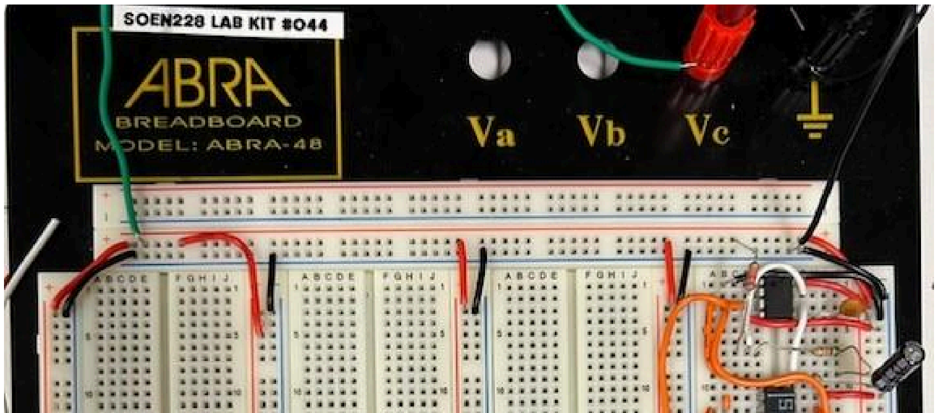
Appendix

Appendix 0 : Completed Project 0



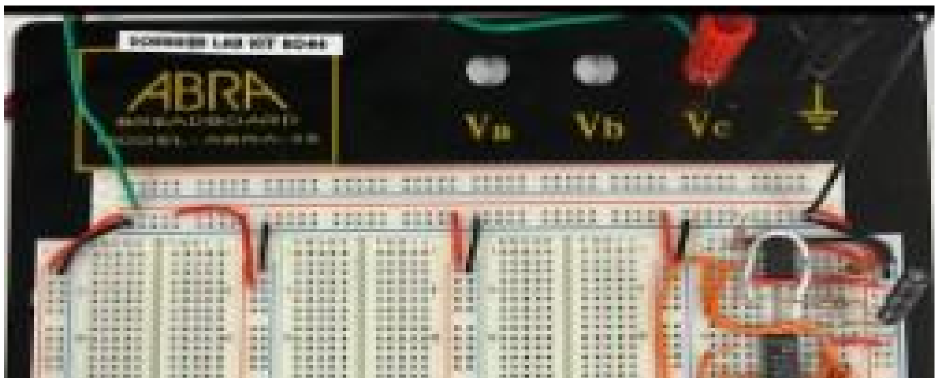
ICs from top to bottom: the 555 timer, to the 74LS164 Shift Register, to the 7420 4 -input NAND gate, followed by the first LED pack.

Appendix 1: Completed Project 1



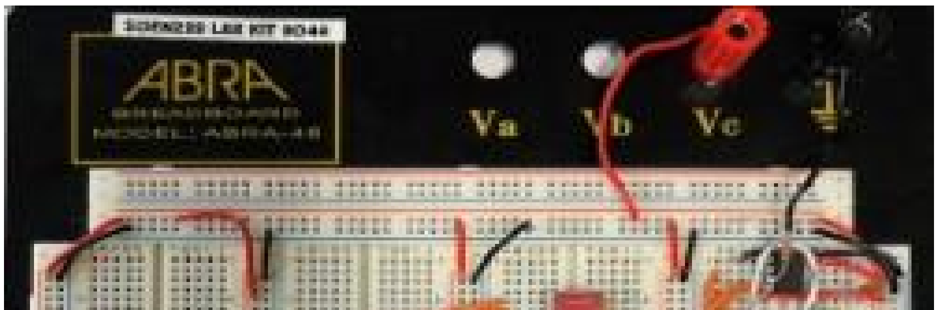
From right to left: the 7404 IC under the first LED lies above the 74LS173 Program Counter, followed by the 4-bit 74LS283 adder, the Sum register and the mirror register which is connected to the second LED (furthest left).

Appendix 2: Completed Project 2



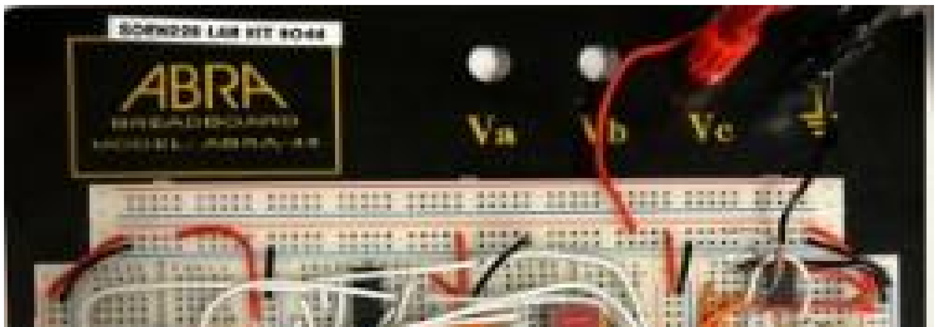
Here, three more 74LS173 ICs are added. Above the sum register, the two data registers are placed one above the other. To the right, the memory address register is connected to the third LED pack of the circuit.

Appendix 3: Completed Project 3



The third LED pack is moved up and the memory address register is now connected to the ATTiny microcontroller and the 7442 decoder, which are placed on top of data register A and B.

Appendix 4: Completed Project 4



In this project, five more ICs are added. Namely, 3 7432 OR gates and 2 7408 AND gates. The AND gates are located on top of the second LED pack on top of one another, followed by the first two OR gates. Lastly, the last OR gate is on top of the 7442 decoder which is connected to the third LED pack. A mini bus of white wires from the first LED pack was made to obtain the timing signals which are inputted into AND and OR gates to operate Data Registers A and B.