Namal University Mianwali

Department of Computer Science

DLD Semester Project (Spring 2022)

Technical Report ***By*** **Muhammad Wasay Tahir**

Contents

Problem Statement: 4

Scope Of This Report: 4

1. Working of Counters (Main & Sub-Counter) 4
2. Implementation of Finite State Machine 4

Working Of Counters: 4

1. Program Counter: 4

1.1. Hardware Implementation of Program Counter: 4

1.2. Schematic Of Program Counter: 5

1.3. Timing Diagram and Truth Table of Program Counter: 5

1.3.1. Timing Diagram of Program Counter: 5

1.3.2. Truth Table of Program Counter: 5

2. Sub-Counter: 6

2.1. Schematic & Timing Diagram of Sub-Counter: 6

Implementation Of Finite State Machine: 6

1. State Control Finite State Machine: 7

1.1. Design Consideration of State Control: 7

1.2. Output Logic of State Control: 7

1.2.1. State Encoding of States: 8

1.2.2. Equations for Output Logic: 8

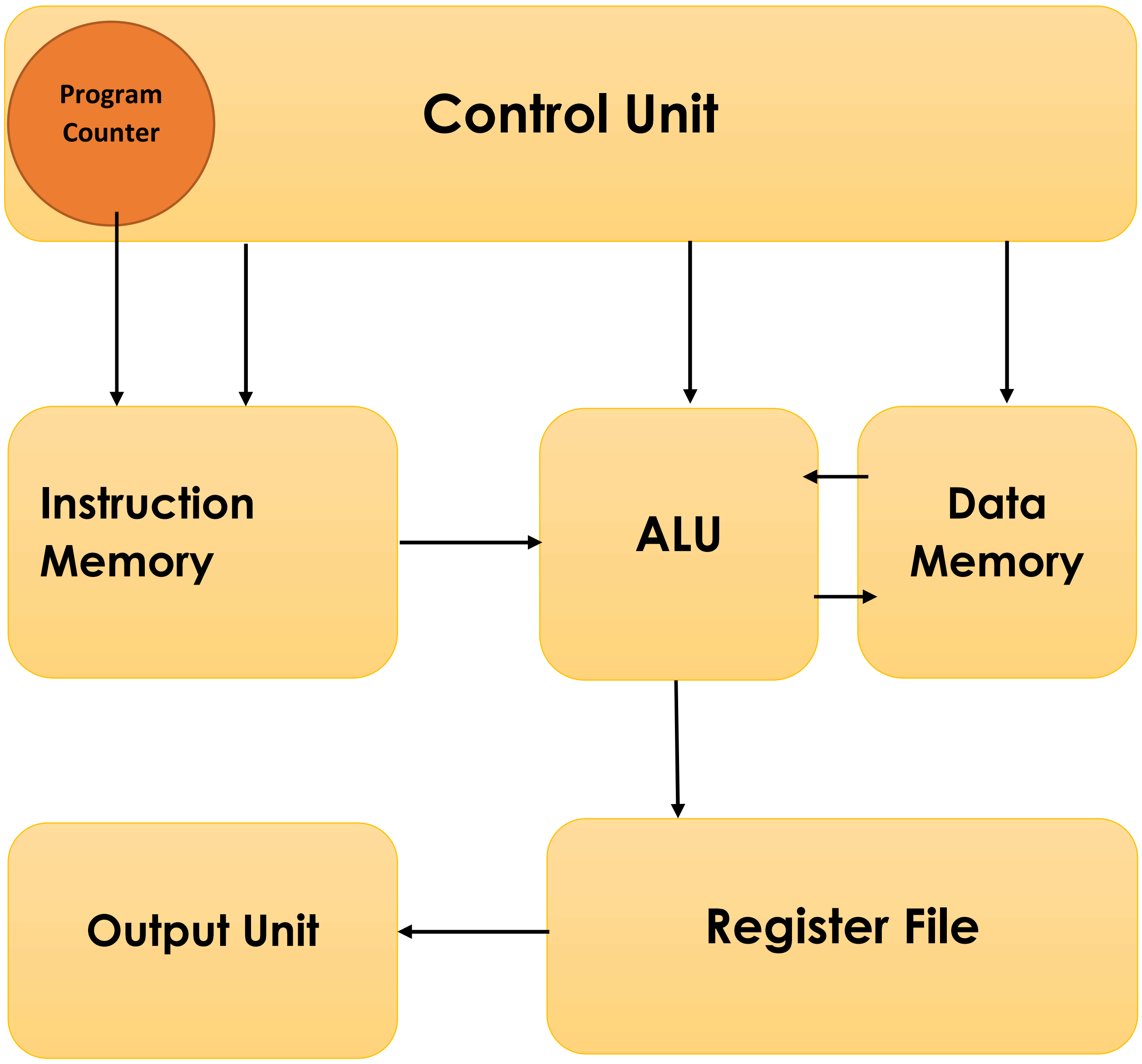
1.2.3. Schematic of Output Logic: 9

1.3. Next-State Logic of State Control: 9

1.4. Application of State Control FSM: 9

1.4.1. Block Diagram: 9
1.4.2. Schematic of State Control FSM: 9
1.4.2.1. Timing Diagram and Execution of Operations: 10
2. Control Unit Finite State Machine: 11
2.1. Truth Table of Control Unit FSM: 11
2.2. Timing Diagram of Control Unit: 11
2.3. Equation of Counter Unit FSM: 11
Hurdles and Hinderance Faced While This Architecture: 12
Limitations: 12
Future Improvements: 12

Block Diagram:



This Report is going to be revolve around the implementation of 4-bit microprocessor. 4-bit integers or other data units are those that are 4 bits wide in computer architecture. In addition, 4-bit central processing unit (CPU) and arithmetic logic unit (ALU) architectures are those based on 4-bit registers or data buses(wires in our case as they carry multiple bit of data). This above drawn is a block diagram our 4-Bit Micro-Architecture, and I'm going to discuss about the Finite State Machines used in my architecture. This report will give you an insight on how this FSM part is the most essential part of the circuit to make this automated. Also this FSM will show their current working state as it'll be iterating by each counter. The block from this is Control Unit And Program Counter.

Problem Statement:

A colleague has asked us to create a prototype programmable computer because we have studied Digital Logic Design and are familiar with design methods. Mr. Malik is unfamiliar with memory management and binary implementation. He requests assistance from our team. He contributes his architecture with us and requests advice and assistance in simulating a microarchitecture. His requirements were to implement a 4-Bit Automated ALU or CPU. This ALU can perform 3 Arithmetic and 3 logical Operators/Functions.

i. Addition
ii. Subtractors
iii. Multiplication
iv. Greater Comparator
v. Equal Comparator
vi. Less Comparator

My contribution to this task is to implement a Finite State Machine to create an Automated Machine that can change states (FSM). I've also created a register file and a display sector to show my current states. In our case, Mr. Malik, the client or user, can also extract saved results from our registers or save results in a specific register.

Scope Of This Report:

This report will focus on the following aspects of our architecture:

1. Working of Counters (Main & Sub-Counter)
2. Implementation of Finite State Machine

Working Of Counters:

I used the built-in counters to generate states for my FSM as well as register addresses. Though you can write a value to each register individually, as a generic condition, I have used the counter values as addresses to that register. In my architecture, I use two types of counters.

1. Program Counter:

The program counter (PC), also known as an instruction pointer (IP), or an instruction counter, or simply a component of the instruction sequencer. A processor register indicates the position of a computer in the program sequence. This program counter generates bits that are used in Instruction Memory and as register file addresses. As this program counter completes a cycle, all of the instructions in the Instruction Memory are executed.

1.1. Hardware Implementation of Program Counter:

This counter generates a sequence of bits within the limit bounded/controlled by me, so it will generate values ranging from 0 to 6, and then reset itself to the initial value.

1.2. Schematic Of Program Counter:

The program counter is built on a three-bit structure. This program counter has only one input, which is a clock, and it produces the bit sequence defined above. One thing to keep in mind is that the clock is set to 13 low ticks and then 1 high tick so that all of the sub-states can be completed without error.

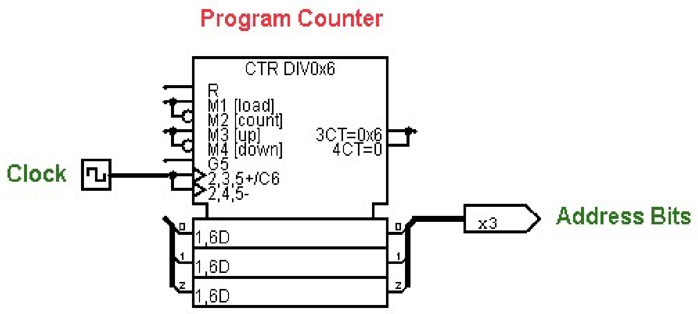


Figure 1

1.3. Timing Diagram and Truth Table of Program Counter:

A time-domain representation of a set of signals is a digital timing diagram. A timing diagram can have several rows, one of which is typically the clock. It is a digital electronics, hardware debugging, and digital communications tool. The digital timing diagram, in addition to providing an overall description of the timing relationships, can aid in the revelation and diagnosis of digital logic. A truth table, on the other hand, is a mathematical table that lists the output of a particular digital logic circuit for all possible input combinations. The logical expression for a specific digital circuit can be calculated using these truth tables.

1.3.1. Timing Diagram of Program Counter:

The timing diagram is a better way of understanding the behavior of a system, and here you can clearly see the working of a program counter. Please refer to the timing diagram below,

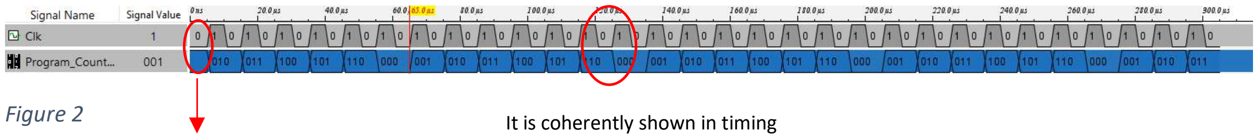


Figure 2

The Counter starts initially from Zero and increment its value on each rising edge of given clock.

It is coherently shown in timing diagram that the value becomes zero after the counter reach 6 or 110 and will again start to increment by 1 from Initial Point i.e 000 in our case.

1.3.2. Truth Table of Program Counter:

State	Value/Address
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110

Table 1

2. Sub-Counter:

The Second Counter is used in a State Controller, which is a sub-controller in each program controller state. This Sub-Counter ranges from 0 to 6 in each Program Counter state and is used in the FSM for circuit automation. This counter's clock is synchronized with the Input Storing Registers, Register File, and Both Rams (Instruction Memory & Data Memory). How this clock is useful in the Register File will be explained later in the report, but Rams and data execution from memory will not be covered in this report.

2.1. Schematic & Timing Diagram of Sub-Counter:

This FSM Sub-Counter is also based on a 3-bit register structure. Because it is bounded to range from 0-6, when the counter reaches its maximum value, it resets itself and returns to the initial stage.

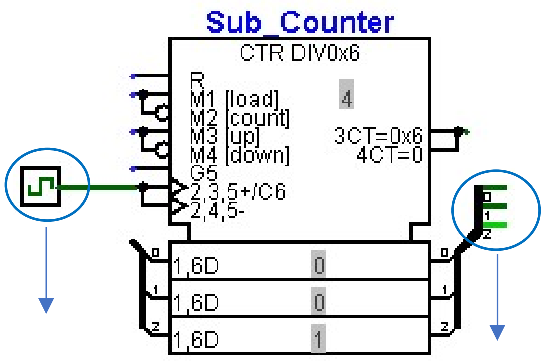


Figure 3

Clock is used to control the FSM's sub-controller. This same clock is being used in several other locations defined above to make this circuit synchronous.

I have used splitter here, the purpose of this splitter is to used single individual bits. This will be briefly explained in the FSM portion

The timing diagram is also attached here to observe the behavior of the counter;

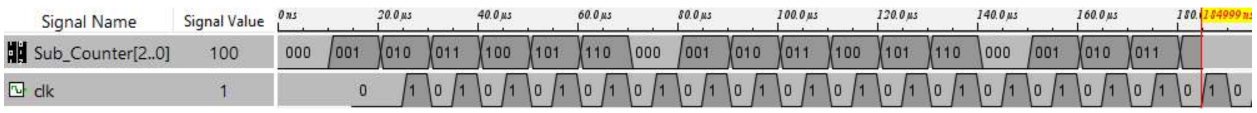


Figure 4

This counter increments its value at every rising edge of the clock and then resets to the initial stage (000) so that the task can continue.

Implementation Of Finite State Machine:

The Finite State Machine is a mathematical abstraction of a sequential logic function. Depending on the purpose, different types of FSM are used. It only has a few inputs, outputs, and states. Flip Flops are used to implement FSMs in real-world circuits. Because the designed FSM is controlled by a Counter in our circuit, it will always follow a sequence on every rising edge of the clock and does not return to a previous state. In my circuit, I used two FSMs, which are discussed briefly below;

I. State Control Machine

II. Control Unit (Simple FSM to Halt and Reset)

1. State Control Finite State Machine:

My primary FSM is the State Control Unit, which is used to repeat certain functions. It will be reiterated in each Program Counter cycle.

1.1. Design Consideration of State Control:

This Finite State Machine has to enable the True or 1 output on different combinations so the specific pins can exhibit the desired functionality.

1. In the initial stage, The FSM has to opaque or 0 all the output pins available in the FSM except the restart pin as I want the circuit to reset the Input containing register (A and B).
2. Then I have to receive or fetch the Data and OP addresses and from the Instruction Memory and also Read the values available on those addresses.
3. Next, I have to store the values somewhere so the Register A is used, and as I know and studied that register only save the values on rising edge of clock and the Enabler Pin should be 1/True in order to write the value.
4. As 1 of the value is stored, now I have to store the second given value. For this the MUX or selection bit have to be 1 (Explained in Design) and the value from the Data Memory has to be read.
5. By loading the value, we have to store it somewhere like we did with value A. The Mux will still be 1 also the Enabler of the Register B should be 1 too. If the Enabler is 0 or False then the value won't be saved.
6. Finally, as all the operation will be performed and we have to store the value in register file, this state will be used to be Idle to handle all the delay and the value of Enabler of register have to be 1 to write this result in Register File.

1.2. Output Logic of State Control:

The considerations made above can be implemented using simple logical combinational logic. Our Output Logic has three bit-width values that are generated by our Next State Logic (Sub-Counter in our case). This logic's Truth Table is explained below *Table 2*;

State	RE_IM	RE_DM	WE_A	WE_B	WE_IM	WE_DM	MUX	WE_FILE	RESET
S0	0	0	0	0	0	0	X	0	1
S1	1	1	0	0	0	0	X	0	0
S2	0	0	1	0	0	0	0	0	0
S3	0	1	0	0	0	0	1	0	0
S4	0	0	0	1	0	0	1	0	0
S5	0	0	0	0	0	0	0	1	0
S6	0	0	0	0	0	0	0	0	0

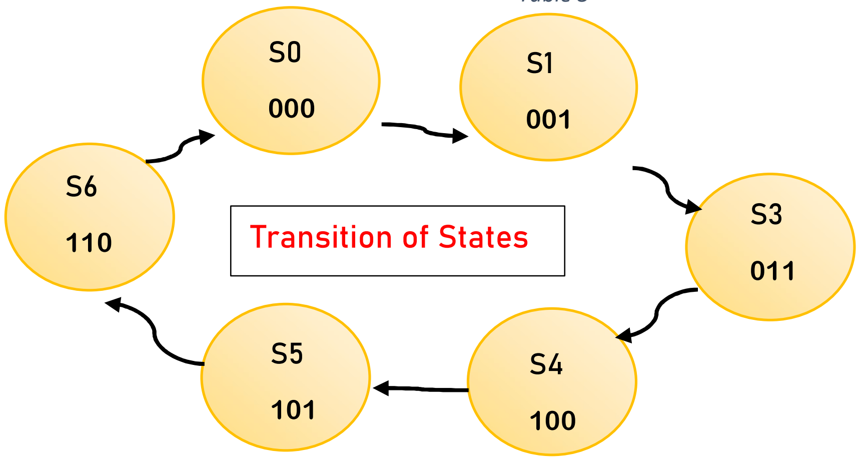
Table 2

1.2.1. State Encoding of States and Transition Table:

The encoding of states is been done below, so my desired function can be achieved in FSM.

STATES	ENCODING
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110

Table 3



1.2.2. Equations for Output Logic:

We can derive the following equations for our output logic using the encoding shown in *Table 3*. Using the logic described in *Table 2*, we'll tally the value for each column and, if 1 is present on any cell, we'll take the encoding of that row for the column name, which is actually our Pins, to automate the process. The equations are defined further below;

RESET = $\overline{S0S1S2}$

RE_IM = S0$\overline{S1S2}$

RE_DM = S0$\overline{S2}$

WE_A = $\overline{S0}$S1$\overline{S2}$

WE_B = $\overline{S0S1}$S2

MUX = $\overline{S0S1}$S2 + S0S1$\overline{S2}$

WE_FILE = S0$\overline{S1}$S2

1.2.3. Schematic of Output Logic:

Each Pin of S2, S1 & S0 is controlled by next logic which determines the state of our program. Please refer to *Figure 5.*

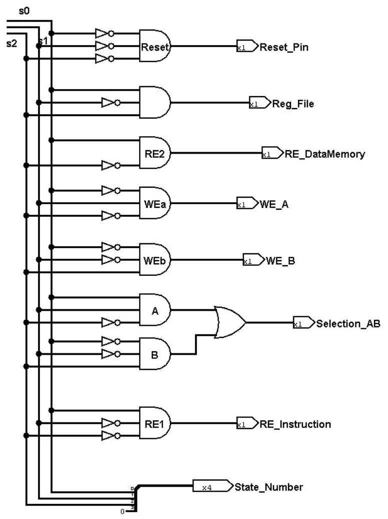


Figure 5

1.3. Next-State Logic of State Control:

Our next state logic is a counter, which I've referred to as a Sub-Counter earlier in the report. This Sub-Counter generates binary values from 0 to 6 in a sequential manner (000-110). Please see *Figures 3* and *Figures 4* for a reference of its schematic and timing diagram.

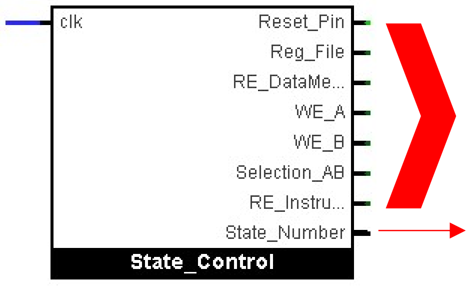
1.4. Application of State Control FSM:

Our State Control Machine will be ready to use after combining the Next-Logic and Output Logic.

1.4.1. Block Diagram:

As shown in *Figures 6*, my state control machine will only receive one input, which will be the clock. This clock is also used in my main circuit (ALU and Registers). The purpose of using the same clock is to ensure that all of my circuits are synchronous. This State Control produces seven results and one state determiner.

The Input Clock Here Is A clock having a 1 high tick followed by 1 low tick



All of these output pins shown here will be combined/attached with our main circuit to help us implement our logic

The State Number Will Output the Current state of our FSM.

Figure 6

1.4.2. Schematic of State Control FSM:

This is the complete working State Control FSM schematic.

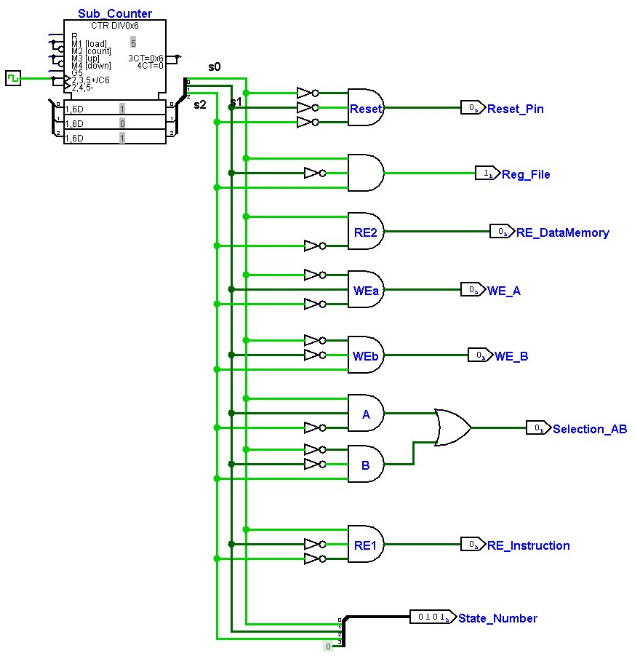


Figure 7

1.4.2.1. Timing Diagram and Execution of Operations:

1.4.2.1.1. Timing Diagram:

As previously stated in this report, a timing diagram is an effective way of inspecting and comprehending our circuit. We can conclude from this that there are only specific signals that are high at every rising edge of the clock, and it will only be high on the rising edge of the clock and specific sequences of Sub-Counter.

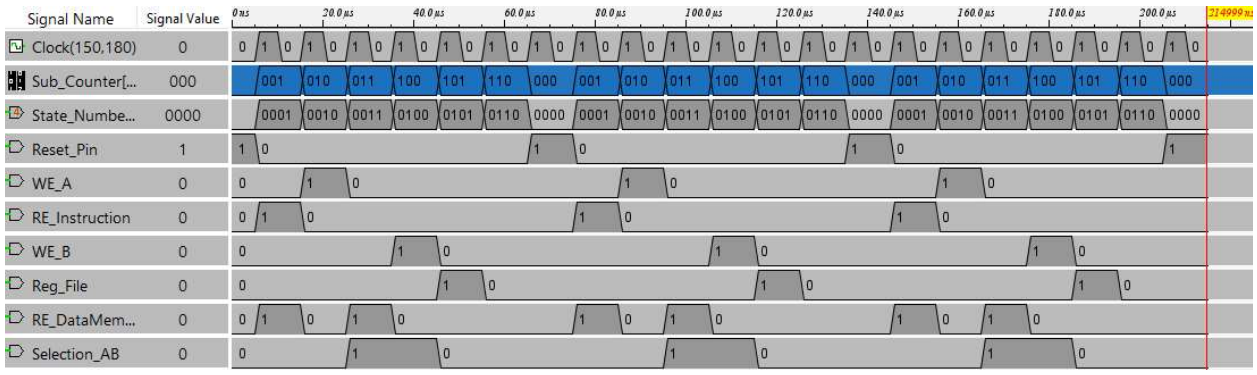


Figure 8

1.4.2.1.2. Execution of Control Signals from State Counter FSM:

To help you understand the FSM, the signals used in *Table 3* and their functionality are defined here.

CONTROL SIGNALS	FUNCTIONALITY
RE_IM	Read Enabler of Instruction Memory
RE_DM	Read Enabler of Data Memory
WE_A	Enabler of Register A
WE_B	Enabler of Register B

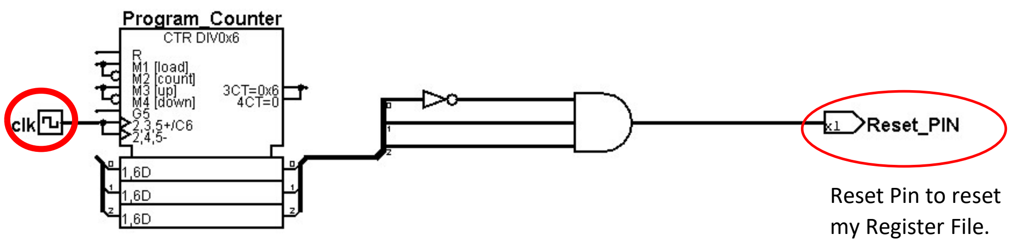
WE_IM	Write Enabler of Instruction Memory
WE_DM	Write Enabler of Data Memory
MUX	Selector of A and B addresses
WE_FILE	Enabler of Register File
RESET	Reset the registers

Table 4

2. Control Unit Finite State Machine:

After retrieving all of the instructions from Instruction Memory, this FSM is used to reset all of the register files. This Counter also has the potential to HALT our architecture. This FSM is relatively simple and straightforward, as it will only return True or 1 when the Program Counter reaches the value of 6.

This clock given here is the clocked at 1 high tick followed by 13 low ticks. This is to make sure that all sub-states have to be completed in its cycle.



Reset Pin to reset my Register File.

Figure 9

2.1. Truth Table of Control Unit FSM:

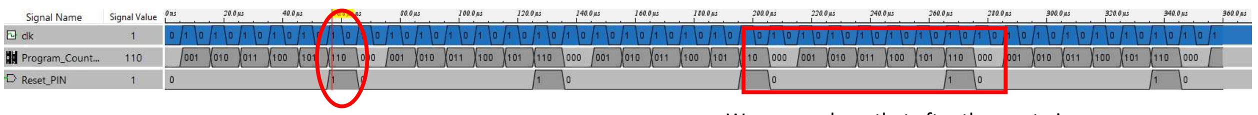
As previously stated, this circuit will only returns a value True/1 when the FSM's Program Counter and Sub-Counter reach their final stages, which is 6(110). This FSM's Truth Table is provided below;

States of Program Counter	Encoding	Reset
S0	000	0
S1	001	0
S2	010	0
S3	011	0
S4	100	0
S5	101	0
S6	110	1

Table 5

2.2. Timing Diagram of Control Unit and State Transition:

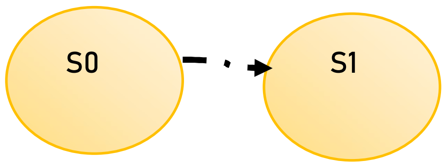
This timing diagram illustrates the logic I created to reset the register file after the program counter has reached its final state. For a better understanding, please see *Figures 10*.



As we can see here that after our program reach its final stage and the rising edge of clock has come, my reset pin is True/1.

Figure 10

We can see here that after the reset pin become 1, it'll maintain its state until the next rising edge came and this will keep recurring if we didn't end/halt our program.



2.3. Equation of Counter Unit FSM:

Table 5, suggests that there is only one state in which Reset Pin is 1. So here is the equation to enable the reset of Counter Unit.

Reset = $\overline{S0}S1S2$

Hurdles and Hinderance Faced While This Architecture:

The main problem I encountered while designing the FSM was that it was not producing the control signals that I expected it to. There was always a one or two clock tick difference. This was a difficult problem to solve. because a minor variation in the clock edge causes the entire operation to fail After numerous failed attempts, I was eventually able to solve this problem.

Also I felt quite circuit while comparing the result of two numbers by a new number. Then I realized there is not much to do in it as I can fed the output of my ALU directly in the Data Memory So That I can perform future operations.

Limitations:

a. The main limitation of our design is that it can only work with 4-bit data. Any calculation involving more than four bits will inevitably overflow. The programmability of our architecture only allows for a limited set of operations. It only displays the value of unsigned operations.
b. Another flaw in our architecture is the requirement to manually write the instructions in the instruction and data memory.
c. Another issue with our design is that when it starts, the circuit retrieves its initial instruction but does not respond. This could be due to the fact that the memories are empty at that point. When it is reset to zero, the same process that produced incorrect results in the previous state now produces accurate calculations.

Future Improvements:

In the future, I'll work to overcome the architecture's short instruction size. Working with a 16- or 32-bit instruction set will be easier and more of 4-bit data. However, putting such a circuit into practice is time-consuming and difficult. Rather than implementing one that performs simple arithmetic operations, work will be done on developing an architecture that can execute and fetch multiple instruction sets in a single clock cycle.