Class Project

The goal for my CSC 720 Class Project was to create a Turing Machine (TM) to perform binary addition and subtraction operations. This machine takes in two binary numbers as input and outputs the result of their addition or subtraction in binary form. By using JFLAP, a finite automata and TM simulation tool, I was able to deepen my understanding of Turing machines and their capabilities. This project also required me to think critically and creatively to design effective solutions and overcome numerous obstacles along the way. I believe my project not only demonstrates my understanding of Turing machines but also shows my perseverance at solving complex problems.

My first draft of my TM calculator quickly became unmanageable with excessive branching for each operation and scenario. Despite my best efforts to stay organized, I found it increasingly difficult to follow and debug. While I was able to create a functioning prototype, I quickly realized there would be no reasonable way to extend my machine's range of operations to include negative results from subtraction. As such, I unfortunately found myself back at square one after many hours of work. Thankfully, I was able to find one solution to combat my challenges of excessive branching and handling negative outputs: additive inverse.

The "additive inverse" property states that for any number x, there exists a number $-x$ that $x + (-x) = 0$. In other words, for every subtraction problem, we can rewrite the second operand as its additive inverse, thus converting the subtraction operation to addition. This property remains true in the case of binary as well, and the additive inverse of a binary number is computed by finding its two's complement. By converting every subtraction problem into an equivalent addition problem, I was able to simplify my machine by eliminating the need for a separate subtraction portion, effectively cutting my machine size in half. Instead, I could do some preprocessing and push all equations through the addition portion.

Furthermore, this property made implementing the capability to perform computations involving negative numbers feasible. Since the initial steps for computing the output of a subtraction operation, whether positive or negative, are identical, I was able to process each function first and not have to worry about determining the positive or negative result until the end. Each scenario begins by calculating the two's complement of the subtrahend and adding the result to the minuend. Thus, removing the requirement for separate calculations resulted in more efficient computations.

As a result of this optimization, the TM is able to not only handle basic arithmetic, but also more complex operations involving negative numbers. In addition to the "additive inverse" property, this capability is made possible by maintaining a consistent input format. To use the machine, the input must be in the format of $\{binary\}\{operation\}\{binary\} = \$$, where $\{binary\}$ represents a binary number, $\{operation\}$ represents either addition or subtraction, and $ is used to notate the end of the input. The resulting output will be displayed between the = and $ characters and padded to the appropriate length based on the size of the largest input binary number. Some examples of correctly formatted input and their corresponding output are shown below:

Table 1: Sample Input and Output for Binary Calculator

INPUT	OUTPUT
$10 + 110 = \$$	$= 1000\$$
$11111 + 011 = \$$	$= 00100010\$$
$1 - 11 = \$$	$= 0010\$$
$00101 - 011 = \$$	$= 00000010\$$

To achieve my goal of creating a fully functional TM calculator that could perform binary addition and subtraction, I divided the building process into six distinct building blocks. Each of these sections focuses on a specific aspect of the computation process, and when combined, work together seamlessly to achieve the desired output. Figure 1 below provides a visual representation and general overview of the entire process and how each block fits into the larger picture. Additionally, a corresponding summary of each section is provided underneath the figure for further clarity.

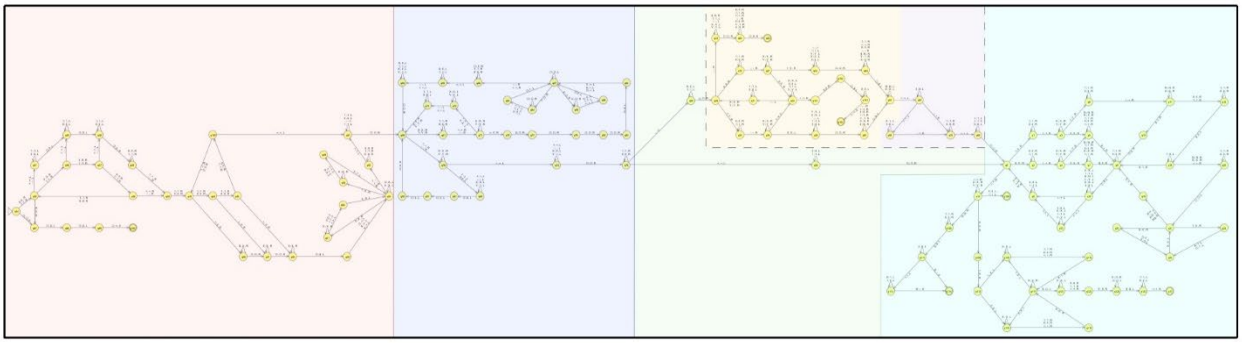

Figure 1: A Breakdown of Turing Machine Building Blocks

1. **<u>Red - Padding:</u>**
    A. Adds leading zeros to each binary input so that its digits are a multiple of four, ensuring consistent length for each input number.
        - $11 \Rightarrow 0011$
        - $10101 \Rightarrow 00010101$

2. **<u>Dark Blue - Length Equality:</u>**
    A. Adds leading zeros to the shorter binary input so that it has the same number of digits as the longer input, allowing for direct comparison of corresponding digits and column-wise addition.
        - $(11110000 - 1100 = \$) \Rightarrow (11110000 - 00001100 = \$)$
        - $(1100 - 11110000 = \$) \Rightarrow (00001100 - 11110000 = \$)$

3. **<u>Green - Determine Operation:</u>**
    A. Determines the operator symbol from the input and routes the equation to the appropriate section for further processing.
        - If the operation is subtraction, the input must go to the 'Yellow - Relational Comparison' section to undergo preprocessing.
        - If the operation is addition, it can forgo preprocessing and continue to the 'Light Blue - Binary Arithmetic' section.

4. **<u>Yellow - Relational Comparison (Subtraction Only):</u>**
    ❖ *Note:* During the arithmetic phase, all problems will appear to the TM as addition, regardless of whether they were input as subtractions. However, it is important to know whether the original operator was subtraction, as it will require post-processing. Furthermore, the post-processing methods will differ depending on whether the result is positive or negative. As such, this section is a two-step process to ensure proper handling

of the arithmetic: identify if the result value will be positive or negative, and notate such so the post-processer knows how to handle the equation.

A. Determines the larger value of the two input numbers, essential for performing subtraction, and determines the sign of the output.
   - If $x > y$, their difference z will be positive: $x - y = z$
   - If $x < y$, their difference z will be negative: $x - y = -z$
   - If $x = y$, their difference z will be 0: $x - y = 0$
     - ❖ *Note: the TM outputs 0 and enters an accept state if this case is met*

B. Notates between "=" sign and the "$" by placing the appropriate flag to allow the TM to apply the appropriate post-processing method
   - If $z > 0$, an S flag is placed to indicate Subtraction was the original operator.
     - $= S\$$
   - If $z < 0$, an N flag is placed to indicate the result will be Negative.
     - $= N\$$

5. <u>Purple - 2s Complement Arithmetic (Subtraction Only):</u>

A. Performs the 2's complement operation on the subtrahend. 2's complement is found by calculating the inverse of the binary value and adding 1.
   - $(0101 - 0011 = S\$) \Rightarrow (0101 - 1100 = S\$) \Rightarrow (0101 - 1101 = S\$)$

B. Changes the subtraction operator to an addition operator.
   - $(0101 - 1101 = S\$) \Rightarrow (0101 + 1101 = S\$)$

6. <u>Light Blue - Binary Arithmetic:</u>

A. Performs binary addition on the two input numbers, from left to right. If the sum of two bits is 10, a 'carry bit' of value 1 is generated and added to the first 0-bit to the left, thus updating it to 1. Any subsequent 1 bits encountered during the iterative process are changed to 0. If there are no 0 bits available to carry over to, the value is extended by one bit to the left. A demonstration of this process is shown below.

$$\begin{array}{cccccccc} 0111 & 0111 & 0111 & 0111 & 0111 & 0111 & 0111 & 0111 \\ \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} & \underline{+\,1011} \\ 1 & 11 & 110 & 100 & 000 & 1000 & 10000 & 10010 \end{array}$$

B. Handles post-processing
   - If result value does not have a flag:
     - Post-processing is not required
     - The value is output
     - TM enters an accept state
   - If result value contains S flag
     - The most significant 1 bit is removed
     - The value is output
     - TM enters an accept state
   - If result value contains N flag
     - The 2s complement for the value is calculated
     - A negative sign is added

- The value is output
- TM enters an accept state

Table 2 below provides an example of the equation's trek across each block as described in the above process. This journey enables the equation to produce the correct output, akin to a functioning calculator.

Table 2: Example Calculation Using Binary Arithmetic Blocks

Input	$11 - 10110 = \$$	
Section	Tape	Description
Padding	$11 - 10110 = \$$	*Tape as it enters section*
	$\mathbf{00}11 - 10110 = \$$	*Pad minuend*
	$0011 - \mathbf{000}10110 = \$$	*Pad subtrahend*
	$0011 - 00010110 = \$$	*Tape as it exits section*
Length Equality	$0011 - 00010110 = \$$	*Tape as it enters section*
	$\mathbf{0000}0011 - 00010110 = \$$	*Add leading zeros to shorter binary input*
	$00000011 - 00010110 = \$$	*Tape as it exits section*
Determine Operation	$00000011 - 00010110 = \$$	*Subtraction Operation, must undergo preprocessing*
Relational Comparison	$00000011 - 00010110 = \$$	*Since $x < y$, their difference z will be negative: $x - y = -z$*
	$00000011 - 00010110 = \mathrm{N}\$$	*As $z < 0$, an N flag is placed to indicate the result will be Negative.*
2s Complement Arithmetic	$00000011 - 00010110 = \mathrm{N}\$$	*Tape as it enters section*
	$00000011 - 11101001 = \mathrm{N}\$$	*Calculate the inverse*
	$00000011 - 11101010 = \mathrm{N}\$$	*Add* 1
	$00000011 + 11101010 = \mathrm{N}\$$	*Change operator*
	$00000011 + 11101010 = \mathrm{N}\$$	*Tape as it exits section*
Binary Arithmetic	$00000011 + 11101010 = \mathrm{N}\$$	*Tape as it enters section*
	$00000011 + 11101010 = \mathrm{N}11101101\$$	*Perform Addition*
	Post processing	
	$= \mathrm{N}11101101\$$	*Determine if Flag used*
	$= \mathrm{N}11101101\$$	*flag N found*
	$= \mathrm{N}11101101\$$	*Find 2s Complement*
	$= \mathrm{N}00010010\$$	*Calculate the inverse*
	$= \mathrm{N}00010011\$$	*Add* 1
	$= -00010011\$$	*Replace N with negative sign*
Output	$= -00010011\$$	

The implementation of this TM has demonstrated its ability to handle a wide range of binary addition and subtraction operations with accuracy. The TM's versatility is apparent as it accepts input in the $\{binary\}\{operation\}\{binary\} = \$$ format, regardless of whether padding or length symmetry is

Input	Output	Result
0000+0011=$	=0011$	Accept
0001+0100=$	=0101$	Accept
0001+1000=$	=1001$	Accept
0010+0111=$	=1001$	Accept
0011+1011=$	=1110$	Accept
	=0$	Accept
10+110=$	=1000$	Accept
0+111=$	=0111$	Accept
1011+11111=$	=00101010$	Accept
111+01010=$	=00010001$	Accept
11111+1=$	=00100000$	Accept
	=0$	Accept
1101-1000=$	=0101$	Accept
1111-0110=$	=1001$	Accept
0110-0001=$	=0101$	Accept
1010-1010=$	=0$	Accept
0111-0011=$	=0100$	Accept
	=0$	Accept
111-11=$	=0100$	Accept
10000-10=$	=00001110$	Accept
1010-111=$	=0011$	Accept
110-0001=$	=0101$	Accept
10-0001=$	=0001$	Accept
	=0$	Accept
0001-1110=$	-1101$	Accept
1010-1111=$	-0101$	Accept
0110-1100=$	-0110$	Accept
0111-1111=$	-1000$	Accept
0010-0100=$	-0010$	Accept
	=0$	Accept
1-11=$	-0010$	Accept
10-1110=$	-1100$	Accept
0001-111=$	-0110$	Accept
10010-111111=$	-00101101$	Accept
01001-1110=$	-00000101$	Accept
0-10=$	-0010$	Accept

Figure 2: Screenshot of JFLAP IO for TM

present. *Figure 2* is a screenshot of JFLAP multiple run (transducer) output and showcases the different types of input the TM can handle, with each section representing specific conditions, including the following:

1. Addition, both digits are multiples of four and do not require padding
2. Addition, both digits are NOT multiples of four and require padding
3. Subtraction, both digits are multiples of four and do not require padding, LHS>RHS so difference is positive
4. Subtraction, both digits are NOT multiples of four and require padding , LHS>RHS so difference is positive
5. Subtraction, both digits are multiples of four and do not require padding, LHS<RHS so difference is negative
6. Subtraction, both digits are NOT multiples of four and require padding , LHS<RHS so difference is negative

As we can see, each input reached an accept state and their corresponding outputs were correct. As such, this TM's ability to perform binary addition and subtraction with a broad range of input types makes it a reliable tool for solving various computational problems.

In conclusion, the creation and implementation of this Turing machine calculator has demonstrated the versatility and accuracy of the Turing machine model in solving computational problems. The machine is able to handle a wide variety of binary addition and subtraction problems without requiring padded or symmetrical inputs, as demonstrated in Figure 2. Furthermore, as this TM was created using JFLAP on a Windows 11 machine, and since JFLAP is a GUI, there is no need for code compilation or running. Anyone can access and install JFLAP from https://www.jflap.org/jflaptmp/ and use it to test and observe this TM's operation. Overall, this project highlights the power and flexibility of the Turing machine model and its potential for solving a complex computational problem.