EAST WEST UNIVERSITY

Project Report

Title: Design a Turing machine using Python, to implement basic operations of TM.

Course Code: CSE360

Course Title: Computer Architecture

Section No: 02

Semester: Summer'21

Submitted By

Name: Md. Abid Hasan Rafe (ID: 2018-3-60-108)

Name: Plabon Banik (2019-1-60-167)

Name: Md. Sabik Alam Rahat (ID: 2019-1-60-256)

Submitted To

Dr. Ahmed Wasif Reza

Associate Professor

Department of CSE, East West University

Date of Submission: 18 Sep, 2021

Introduction: Turing machine invented by the British mathematician Alan Turing in the early 1900s, Turing Machines were originally only built to define the word "algorithm" a great challenge of the 20th century, laid out by the mathematician David Hilbert in his address to the International Congress of Mathematicians in 1900i. However, since then, Turing Machines have evolved to become one of the simplest models of the modern computer – that is to say, whatever modern computers can do, Turing Machines can do and whatever Turing Machines can do, modern computers can do. They are equivalent. This theory of equivalence between Turing Machines and modern computers is now part of the Church-Turing thesis. Since the entire program is based on the theory of Turing Machine, we will describe this theory at length in the introduction and refer to it later in the report. Turing Machines consist of a tape limited on one side but infinite on the other (semi-finite tape) and a read/write head (see figure 1).

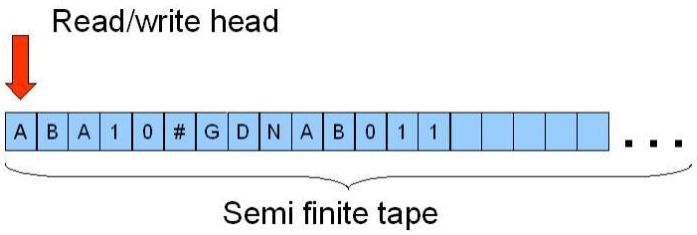


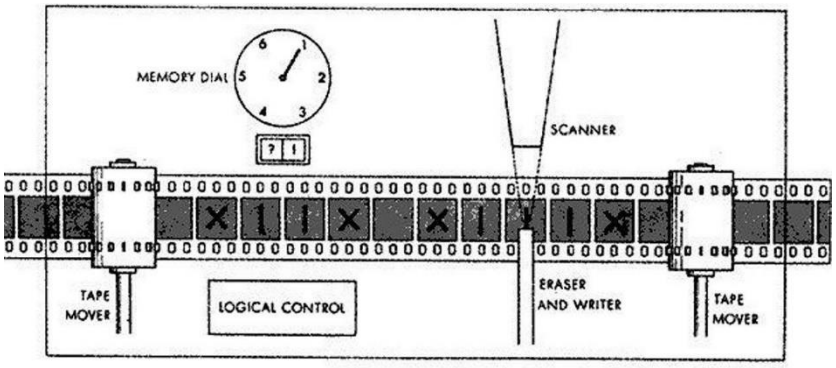
Figure 1 – The Turing Machine

Literature Review: A Turing machine is a mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, given any computer algorithm, a Turing machine capable of simulating that algorithm's logic can be constructed. The purpose of this report is to show how we designed the Turing Machine (TM) using python and how we implemented the basic operations of mathematics (addition, subtraction, divisions and multiplications) using this code. This report also shows the results and analysis done during the time of making of the project.

The tape is divided into cells and each cell can contain one symbol only. However, a symbol could be more than one character (for example big Symbol could be a symbol). A special symbol, the blank symbol, is used to fill every empty cell. The read head can move left and right anywhere over the tape, one cell at the time. When it is immediately over a cell, it can read the symbol there, write a new symbol (overwriting the one already there) or simply do nothing.

At any one time, the machine has a head which is positioned over one of the squares on the tape. With this head, the machine can perform three very basic operations:

1. Read the symbol on the square under the head.
2. Edit the symbol by writing a new symbol or erasing it.
3. Move the tape left of right by one square so that the machine can read and edit the symbol on a neighboring square.



Problem Statements & Objects: The machine to emulate a computer would contain 4 tapes. One tape for user defined variables, one tape for system variables, one tape for the call stack and one tape for processing. Before compilation to a Turing Machine, the source code would have to be parsed and converted to a tree, which would allow the entire code to be executed by the following four functions only:

- Addition
- Subtraction
- Multiplication
- Division

Addition: Addition is extremely simple. Assuming numbers are given in a unary format, all that needs to be done is the copying of the second number to the first number, using a simple copying machine. In the context of the architecture of our multiple-tape machine, two new system variables could be created on the system variables tape, one with each number (see sections on variables below). The copying machine would then concatenate both variables.

Subtraction: Subtraction is somewhat more complicated to implement. Let us take a – b as an example. The simplest approach is to take both the numbers in unary format and to cross a 1 from a for every 1 from b. The resulting number left in a will be the result of the subtraction. If negative subtraction also needs to be supported, the Machine could check, every time it crossed a 1 from a whether a was now equal to 0. If that was the case, it could write a – sign at the start of the tape and add 1s instead of removing them thereafter.

Multiplication: Multiplication could also be achieved using a simple copying machine. The number, again in unary format, would be copied into a result variable repeatedly until the end of the multiplication. For example, in the multiplication 5 × 3, the unary number 11111 would be repeated three times into the result variable.

Division: Division is also relatively complicated to implement. The two numbers would, again, must be stored in unary format in two system variables. Then, the machine would repeatedly try to remove a copy of the second number from the first, keeping count of the number of removals. For example, for the division 10 ÷ 5 (coded 1111111111 ÷ 11111), the program would repeatedly try to remove 5 1s (11111 – the encoding for the number 5) from the encoding for 10 until it ran out. When it runs out, the number of removals carried out is equal to the result of the division.

• If they match, they are deleted, and the entire process starts again from the next characters (which are now the first characters of the string)

• If they do not match, the equation should return false immediately Once either string runs out, the machine would then check the status of the other string – if it ran out at the same time, the machine would be accepted – otherwise, rejected.

Proposed Method & Design:

- In one move the TM will:
  - Change state, which may be the same as the current state
  - Write a tape symbol in the current cell, which may be the same as the current symbol
  - Move the tape head left or right one cell
  - The special states for rejecting and accepting take effect immediately
- Formally, the Turing Machine is denoted by the 8-tuple:
  - $M = (Q, \square, \Gamma, \delta, q, B, q, q)$
    $0 \quad a \quad r$

- Q = finite states of the control
- □ = finite set of input symbols, which is a subset of Γ below
- Γ = finite set of tape symbols
- δ = transition function. δ (q, X) are a state and tape symbol X.
  - The output is the triple, (p, Y, D)

– Where p = next state, Y = new symbol written on the tape, D = direction to move the tape head

- q_0 = start state for finite control
- B = blank symbol. This symbol is in Γ but not in □.
- q_{accept} = set of final or accepting states of Q.
- q_{reject}

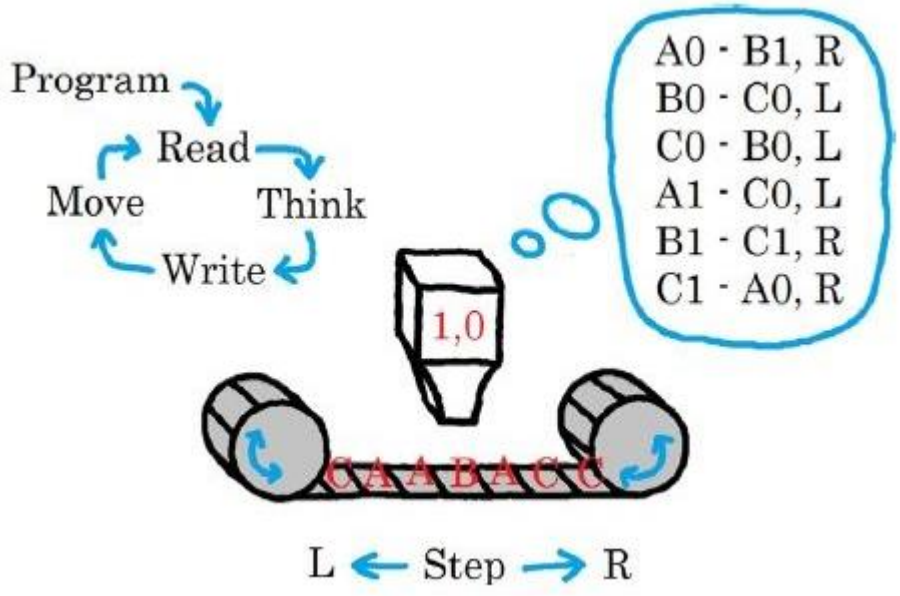
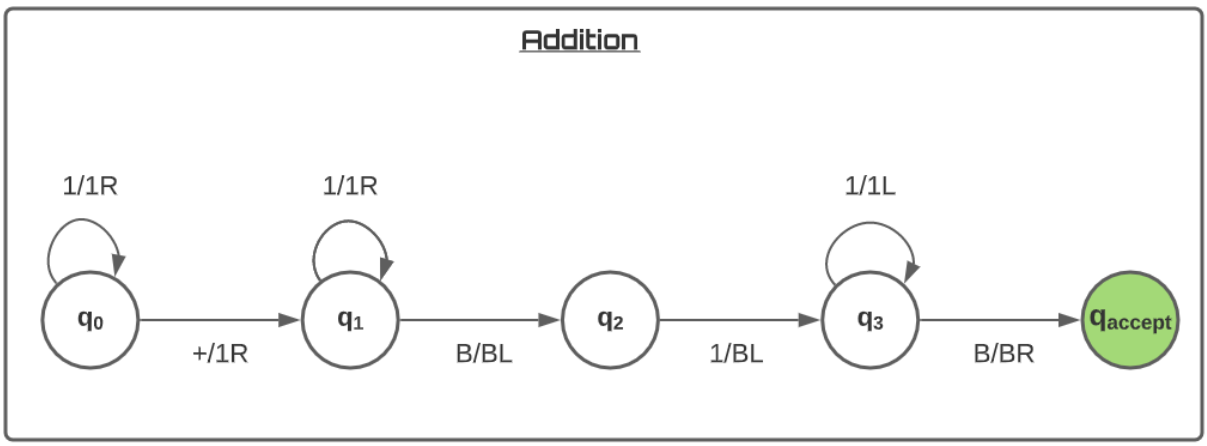


Figure: Turing Machine Architecture

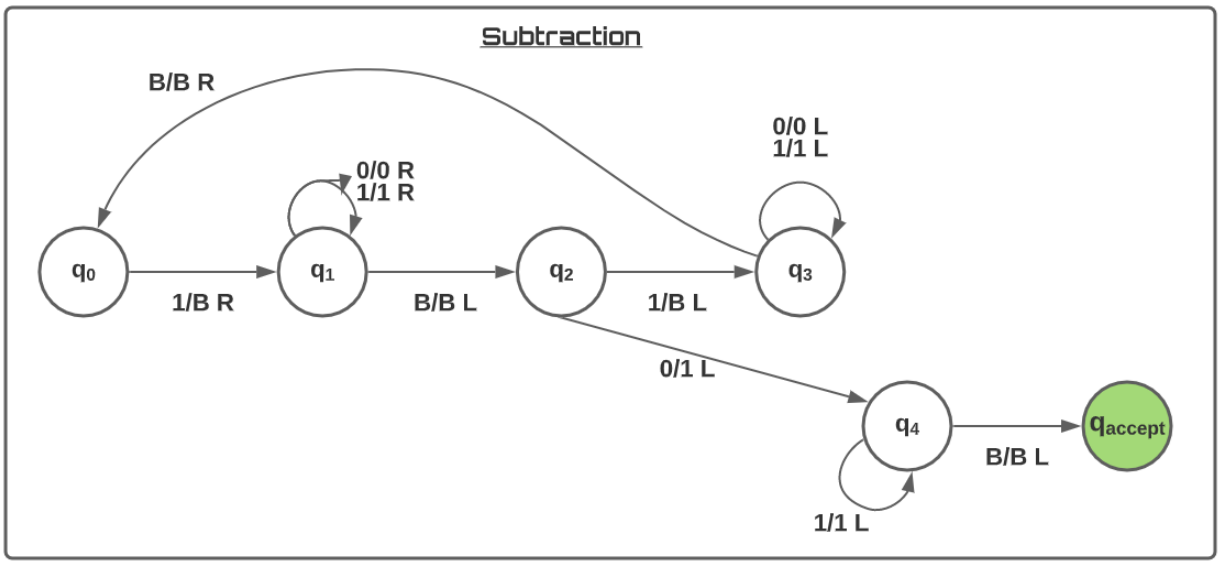
We have designed state transition diagram for addition as follows:

Addition:



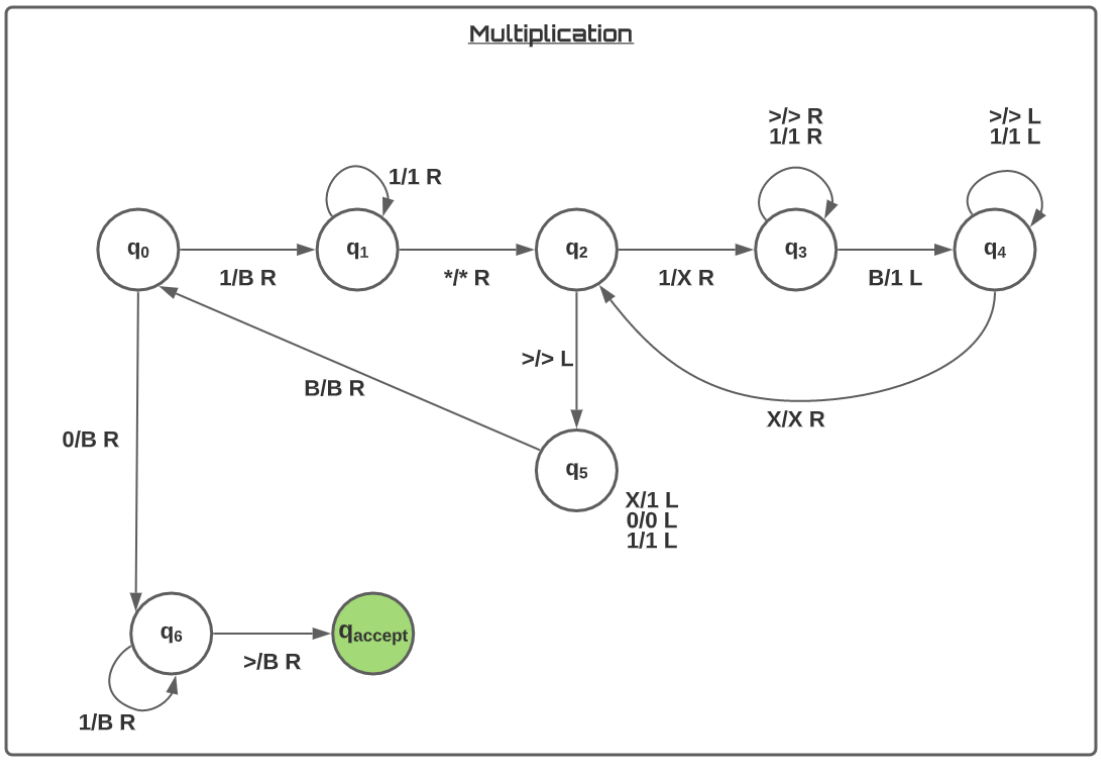
```
tm = TuringMachine(states={'A', 'B', 'C', 'D', 'Accept'},
    symbols={'0', '1'},
    blank_symbol="\033[31m~\033[0m",
    input_symbols={'1'},
    initial_state='A',
    accepting_states={'Accept'},
    transitions={
        ('A', '1'): ('A', '1', 1),
        ('A', '+'): ('B', '1', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', "\033[31m~\033[0m"): ('C', "\033[31m~\033[0m", -1),
        ('C', '1'): ('D', "\033[31m~\033[0m", -1),
        ('D', '1'): ('D', '1', -1),
        ('D', "\033[31m~\033[0m"): ('Accept', "\033[31m~\033[0m", 1),
})
```

Subtraction:



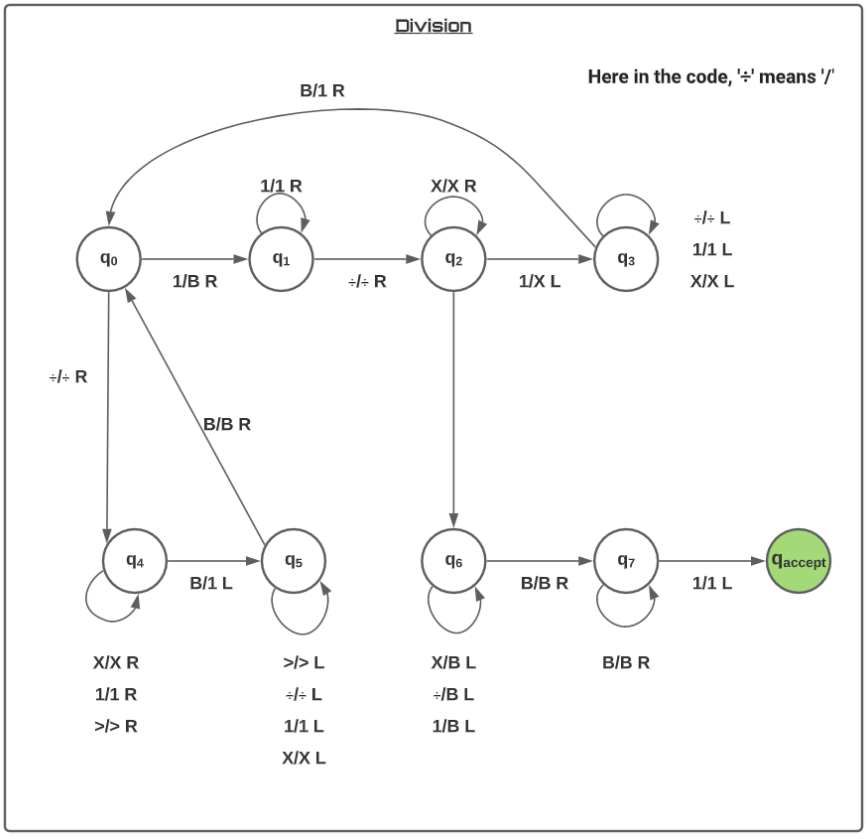
```
tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'Accept'},
    symbols={'0', '1'},
    blank_symbol='\033[31m~\033[0m',
    input_symbols={'1'},
    initial_state='A',
    accepting_states={'Accept'},
    transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', '-'): ('B', '-', 1),
        ('B', '\033[31m~\033[0m'): ('C', '\033[31m~\033[0m', -1),
        ('C', '1'): ('D', '\033[31m~\033[0m', -1),
        ('C', '-'): ('E', '1', -1),
        ('D', '-'): ('D', '-', -1),
        ('D', '1'): ('D', '1', -1),
        ('D', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('E', '1'): ('E', '1', -1),
        ('E', '\033[31m~\033[0m'): ('Accept', '\033[31m~\033[0m', 1),
})
```

Multiplication:



```
tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'F', 'G' 'Accept'},
    symbols={'0', '1'},
    blank_symbol='\033[31m~\033[0m',
    input_symbols={'1'},
    initial_state='A',
    accepting_states={'Accept'},
    transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', '*'): ('C', '*', 1),
        ('C', '1'): ('D', 'x', 1),
        ('D', '>'): ('D', '>', 1),
        ('D', '1'): ('D', '1', 1),
        ('D', '\033[31m~\033[0m'): ('E', '1', -1),
        ('E', '1'): ('E', '1', -1),
        ('E', '>'): ('E', '>', -1),
        ('E', 'x'): ('C', 'x', 1),
        ('C', '>'): ('F', '>', -1),
        ('F', 'x'): ('F', '1', -1),
        ('F', '*'): ('F', '*', -1),
        ('F', '1'): ('F', '1', -1),
        ('F', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('A', '*'): ('G', '\033[31m~\033[0m', 1),
        ('G', '1'): ('G', '\033[31m~\033[0m', 1),
        ('G', '>'): ('Accept', '\033[31m~\033[0m', 1),
})
```

Division:



```
tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'F', 'G', 'I', 'Accept'},
    symbols={'0', '1'},
    blank_symbol='\033[31m~\033[0m',
    input_symbols={'1'},
    initial_state='A',
    accepting_states={'Accept'},
    transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', '/'): ('C', '/', 1),
        ('C', '1'): ('D', 'x', -1),
        ('D', '/'): ('D', '/', -1),
        ('D', '1'): ('D', '1', -1),
        ('D', '\033[31m~\033[0m'): ('A', '1', 1),
        ('C', 'x'): ('C', 'x', 1),
        ('D', 'x'): ('D', 'x', -1),
        ('A', '/'): ('E', '/', 1),
        ('E', 'x'): ('E', 'x', 1),
        ('E', '1'): ('E', '1', 1),
        ('E', '>'): ('E', '>', 1),
        ('E', '\033[31m~\033[0m'): ('F', '1', -1),
        ('F', '>'): ('F', '>', -1),
        ('F', '/'): ('F', '/', -1),
        ('F', '1'): ('F', '1', -1),
        ('F', 'x'): ('F', 'x', -1),
        ('F', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('C', '>'): ('G', '\033[31m~\033[0m', -1),
        ('G', 'x'): ('G', '\033[31m~\033[0m', -1),
        ('G', '/'): ('G', '\033[31m~\033[0m', -1),
        ('G', '1'): ('G', '\033[31m~\033[0m', -1),
        ('G', '\033[31m~\033[0m'): ('I', '\033[31m~\033[0m', 1),
        ('I', '\033[31m~\033[0m'): ('I', '\033[31m~\033[0m', 1),
        ('I', '1'): ('Accept', '1', -1),
})
```

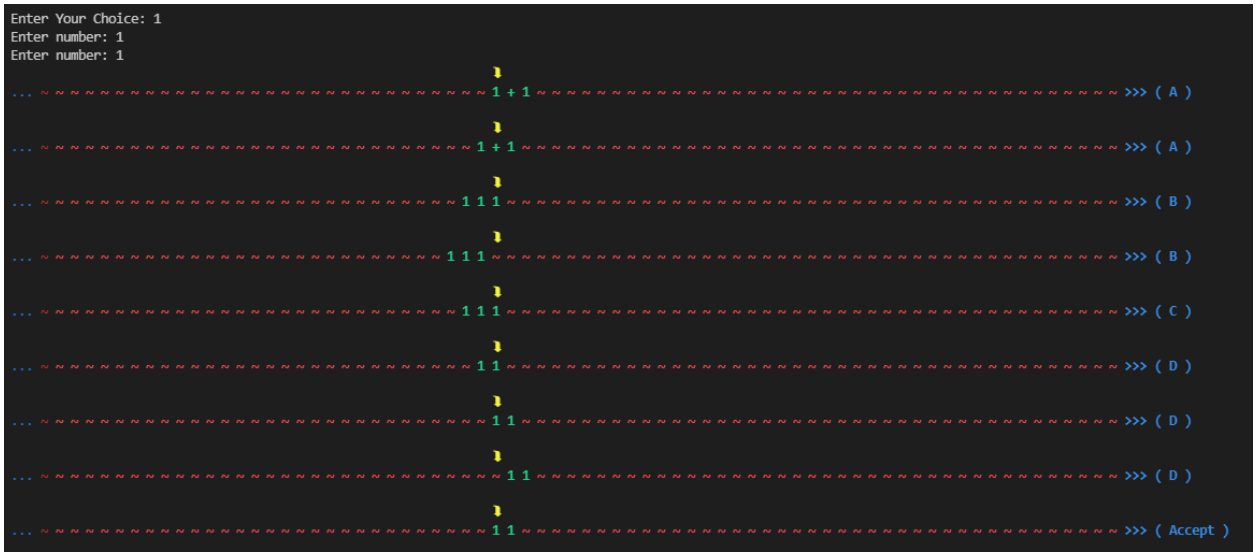
Results Analysis:

User Interface:

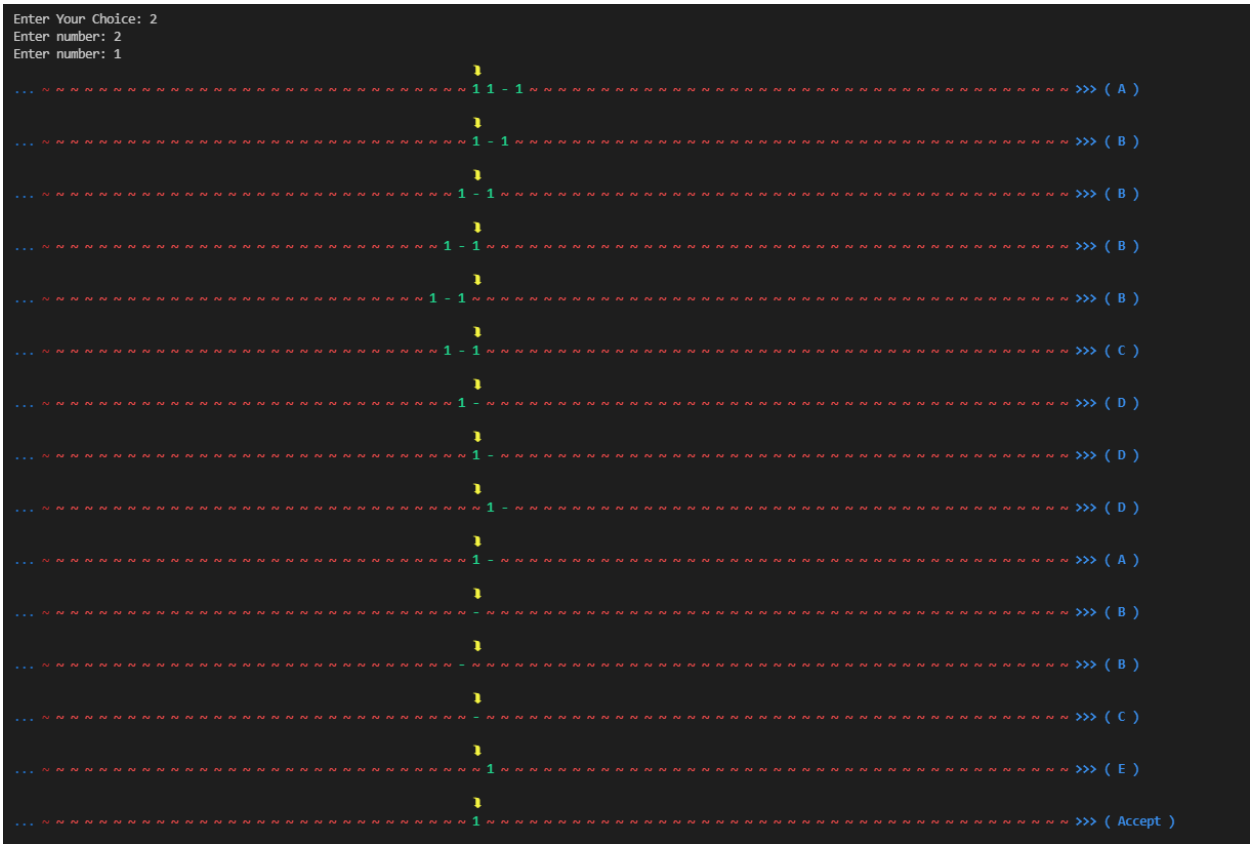
```
PS C:\Users\sabik> & C:/Users/sabik/AppData/Local/Programs/Python/Pyt
*-----------------------------------------*
*-----------------------------------------*
|            Turning Machine              |
*-----------------------------------------*
*-----------------------------------------*
|       1. Addition of Two Number         |
|-----------------------------------------|
|-----------------------------------------|
|       2. Subtraction of Two Number      |
|-----------------------------------------|
|-----------------------------------------|
|       3. Multiplication of Two Number   |
|-----------------------------------------|
|-----------------------------------------|
|       4. Division of Two Number         |
|-----------------------------------------|
|-----------------------------------------|
|       5. Exit                           |
*-----------------------------------------*
*-----------------------------------------*


Enter Your Choice:
```

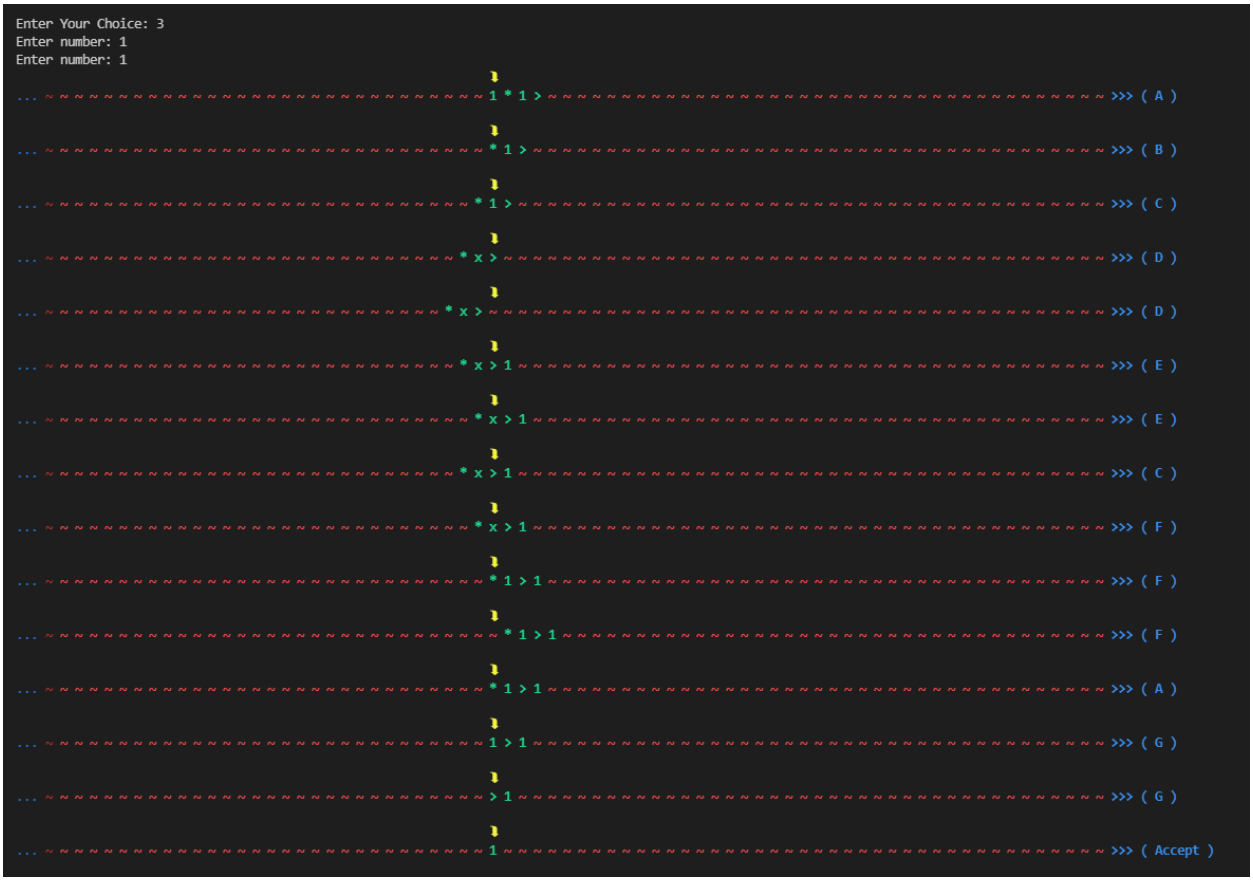
Addition:



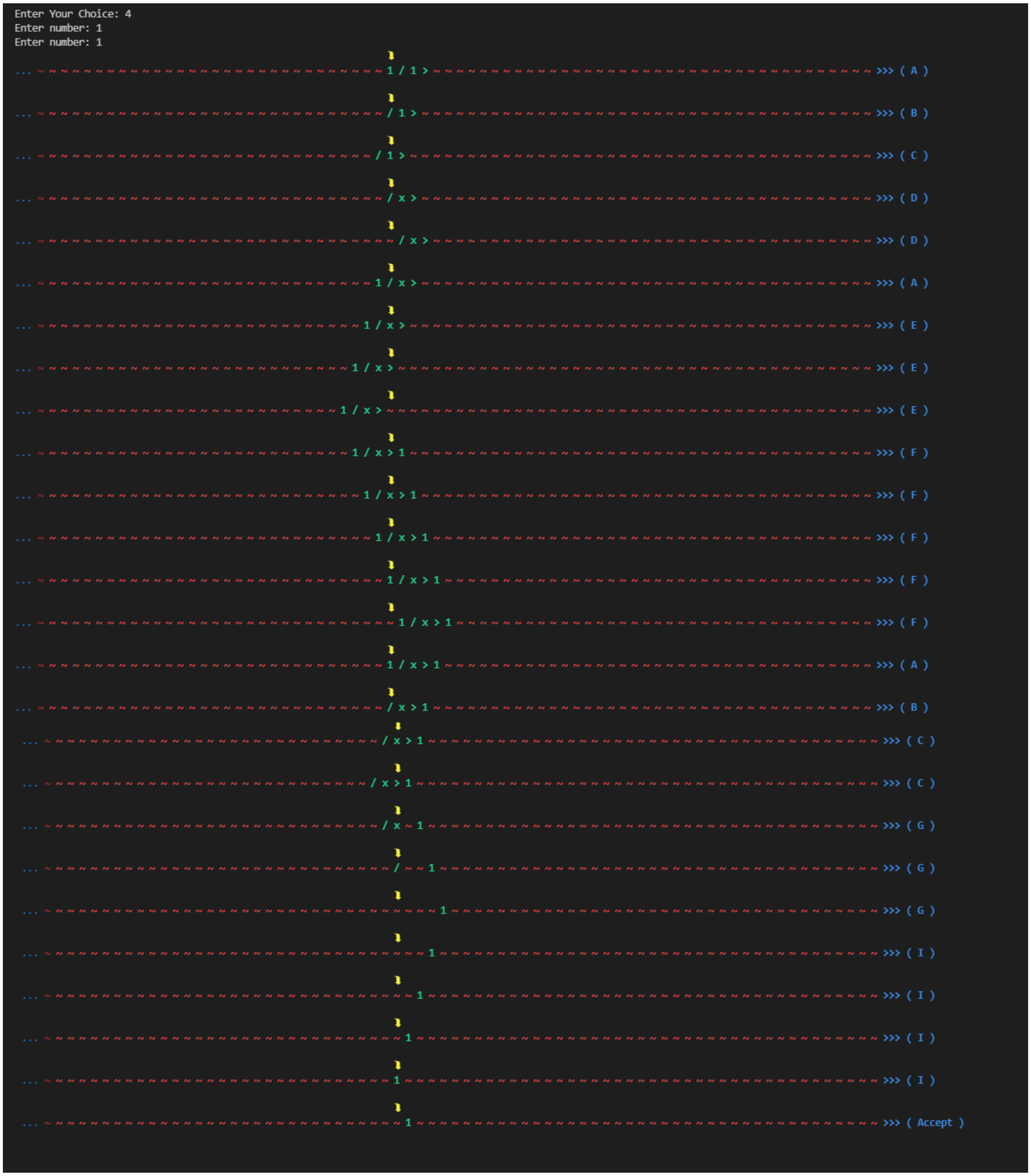
Subtraction:



Multiplication:



Division:



Exit:

Summary of Findings:

We took help from Google, GitHub, and YouTube etc. We also took help from some research paper based on Turing Machine. These papers helped us a lot to understand the main concept of this Turing Machine. It also encourages us to work on it in future.

Future work:

- Equation
- Square
- Check Palindrome
- Check Parity
- Check even/odd

We will implement these functions in future.

Limitations:

- Only able to solve the basic mathematical operations as:
- Our function can’t add any double or floating number.
- Our function can’t calculate negative value.
- In division function complexity can’t be handle.

We will try our best to solve these limitations in future.

Appendix:

```
import time
from collections import defaultdict
from dataclasses import dataclass, field


@dataclass
class TuringMachine:
    states: set[str]
    symbols: set[str]
    blank_symbol: str
    input_symbols: set[str]
    initial_state: str
    accepting_states: set[str]
    transitions: dict[tuple[str, str], tuple[str, str, int]]

    head: int = field(init=False)
    tape: defaultdict[int, str] = field(init=False)
    current_state: str = field(init=False)
    halted: bool = field(init=False, default=True)

    def initialize(self, input_symbols):
        self.head = 0
        self.halted = False
        self.current_state = self.initial_state
        self.tape = defaultdict(lambda: "\033[31m~\033[0m", input_symbols)

    def step(self):
        if self.halted:
            raise RuntimeError('Cannot step halted machine')

        try:
            state, symbol, direction = self.transitions[(
                self.current_state, self.tape[self.head])]
        except KeyError:
            self.halted = True
            return
        self.tape[self.head] = symbol
        self.current_state = state
        self.head += direction

    def accepted_input(self):
        if not self.halted:
            raise RuntimeError('Machine still running')
        return self.current_state in self.accepting_states

    def print(self, window=30):
        print(f'{" " * (2 * window + 4)}\033[1;33m⬇\033[0m')
        print('\033[34m... \033[0m', end='')
        print("\033[1;32m ".join(self.tape[i] for i in range(
            self.head - window, self.head + window+12)), end=''"\033[0m")
```

```
        print(f'\033[1;34m >>> ( {self.current_state} )\033[0m\n')


def split(word):
    return [char for char in word]


def addition():
    tm = TuringMachine(states={'A', 'B', 'C', 'D', 'Accept'},
                       symbols={'0', '1'},
                       blank_symbol="\033[31m~\033[0m",
                       input_symbols={'1'},
                       initial_state='A',
                       accepting_states={'Accept'},
                       transitions={
        ('A', '1'): ('A', '1', 1),
        ('A', '+'): ('B', '1', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', "\033[31m~\033[0m"): ('C', "\033[31m~\033[0m", -1),
        ('C', '1'): ('D', "\033[31m~\033[0m", -1),
        ('D', '1'): ('D', '1', -1),
        ('D', "\033[31m~\033[0m"): ('Accept', "\033[31m~\033[0m", 1),
    })

    num1 = int(input("Enter number: "))
    num2 = int(input("Enter number: "))

    str1 = ("1"*num1)+"+"
    str2 = ("1"*num2)

    unaryinput = str1+str2
    temp = (split(unaryinput))
    res = dict()
    for idx, ele in enumerate(temp):
        res[idx] = ele

    tm.initialize(res)

    while not tm.halted:
        tm.print()
        tm.step()


def subtraction():
    tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'Accept'},
                       symbols={'0', '1'},
                       blank_symbol='\033[31m~\033[0m',
                       input_symbols={'1'},
                       initial_state='A',
                       accepting_states={'Accept'},
                       transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
```

```
        ('B', '-'): ('B', '-', 1),
        ('B', '\033[31m~\033[0m'): ('C', '\033[31m~\033[0m', -1),
        ('C', '1'): ('D', '\033[31m~\033[0m', -1),
        ('C', '-'): ('E', '1', -1),
        ('D', '-'): ('D', '-', -1),
        ('D', '1'): ('D', '1', -1),
        ('D', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('E', '1'): ('E', '1', -1),
        ('E', '\033[31m~\033[0m'): ('Accept', '\033[31m~\033[0m', 1),
    })

    num1 = int(input("Enter number: "))
    num2 = int(input("Enter number: "))
    mx = max(num1, num2)
    mn = min(num1, num2)

    str1 = ("1"*mx)+"-"
    str2 = ("1"*mn)

    unaryinput = str1+str2
    temp = (split(unaryinput))
    res = dict()
    for idx, ele in enumerate(temp):
        res[idx] = ele

    tm.initialize(res)

    while not tm.halted:
        tm.print()
        tm.step()


def multiplication():
    tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'F', 'G' 'Accept'},
                       symbols={'0', '1'},
                       blank_symbol='\033[31m~\033[0m',
                       input_symbols={'1'},
                       initial_state='A',
                       accepting_states={'Accept'},
                       transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', '*'): ('C', '*', 1),
        ('C', '1'): ('D', 'x', 1),
        ('D', '>'): ('D', '>', 1),
        ('D', '1'): ('D', '1', 1),
        ('D', '\033[31m~\033[0m'): ('E', '1', -1),
        ('E', '1'): ('E', '1', -1),
        ('E', '>'): ('E', '>', -1),
        ('E', 'x'): ('C', 'x', 1),
        ('C', '>'): ('F', '>', -1),
        ('F', 'x'): ('F', '1', -1),
        ('F', '*'): ('F', '*', -1),
```

```
        ('F', '1'): ('F', '1', -1),
        ('F', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('A', '*'): ('G', '\033[31m~\033[0m', 1),
        ('G', '1'): ('G', '\033[31m~\033[0m', 1),
        ('G', '>'): ('Accept', '\033[31m~\033[0m', 1),
    })

    num1 = int(input("Enter number: "))
    num2 = int(input("Enter number: "))
    mx = max(num1, num2)
    mn = min(num1, num2)

    str1 = ("1"*mn)+"*"
    str2 = ("1"*mx)+">"

    unaryinput = str1+str2
    temp = (split(unaryinput))
    res = dict()
    for idx, ele in enumerate(temp):
        res[idx] = ele

    tm.initialize(res)

    while not tm.halted:
        tm.print()
        tm.step()


def division():
    tm = TuringMachine(states={'A', 'B', 'C', 'D', 'E', 'F', 'G', 'I',
'Accept'},
                       symbols={'0', '1'},
                       blank_symbol='\033[31m~\033[0m',
                       input_symbols={'1'},
                       initial_state='A',
                       accepting_states={'Accept'},
                       transitions={
        ('A', '1'): ('B', '\033[31m~\033[0m', 1),
        ('B', '1'): ('B', '1', 1),
        ('B', '/'): ('C', '/', 1),
        ('C', '1'): ('D', 'x', -1),
        ('D', '/'): ('D', '/', -1),
        ('D', '1'): ('D', '1', -1),
        ('D', '\033[31m~\033[0m'): ('A', '1', 1),
        ('C', 'x'): ('C', 'x', 1),
        ('D', 'x'): ('D', 'x', -1),
        ('A', '/'): ('E', '/', 1),
        ('E', 'x'): ('E', 'x', 1),
        ('E', '1'): ('E', '1', 1),
        ('E', '>'): ('E', '>', 1),
        ('E', '\033[31m~\033[0m'): ('F', '1', -1),
        ('F', '>'): ('F', '>', -1),
        ('F', '/'): ('F', '/', -1),
```

```
        ('F', '1'): ('F', '1', -1),
        ('F', 'x'): ('F', 'x', -1),
        ('F', '\033[31m~\033[0m'): ('A', '\033[31m~\033[0m', 1),
        ('C', '>'): ('G', '\033[31m~\033[0m', -1),
        ('G', 'x'): ('G', '\033[31m~\033[0m', -1),
        ('G', '/'): ('G', '\033[31m~\033[0m', -1),
        ('G', '1'): ('G', '\033[31m~\033[0m', -1),
        ('G', '\033[31m~\033[0m'): ('I', '\033[31m~\033[0m', 1),
        ('I', '\033[31m~\033[0m'): ('I', '\033[31m~\033[0m', 1),
        ('I', '1'): ('Accept', '1', -1),
    })

    num1 = int(input("Enter number: "))
    num2 = int(input("Enter number: "))
    mx = max(num1, num2)
    mn = min(num1, num2)

    str1 = ("1"*mn)+"/"
    str2 = ("1"*mx)+">"

    unaryinput = str1+str2
    temp = (split(unaryinput))
    res = dict()
    for idx, ele in enumerate(temp):
        res[idx] = ele

    tm.initialize(res)

    while not tm.halted:
        tm.print()
        tm.step()


if __name__ == '__main__':

    while True:
        print("*-----------------------------------------*")
        print("*-----------------------------------------*")
        print("|             Turning Machine             |")
        print("*-----------------------------------------*")
        print("*-----------------------------------------*")
        print("|        1. Addition of Two Number        |")
        print("|-----------------------------------------|")
        print("|-----------------------------------------|")
        print("|        2. Subtraction of Two Number     |")
        print("|-----------------------------------------|")
        print("|-----------------------------------------|")
        print("|        3. Multiplication of Two Number  |")
        print("|-----------------------------------------|")
        print("|-----------------------------------------|")
        print("|        4. Division of Two Number        |")
        print("|-----------------------------------------|")
        print("|-----------------------------------------|")
```

```
        print("|        5. Exit                                |")
        print("*----------------------------------------------*")
        print("*----------------------------------------------*")

        choice = int(input("\n\nEnter Your Choice: "))
        if(choice == 1):
            addition()
        if(choice == 2):
            subtraction()
        if(choice == 3):
            multiplication()
        if(choice == 4):
            division()
        if(choice == 5):
            print("*----------------------------------------------*")
            print("*----------------------------------------------*")
            print("|     Thank You for Using Out System        |")
            print("*----------------------------------------------*")
            print("*----------------------------------------------*")
            break
```

Conclusion: Turing is a machining process in which a cutting tool, typically a non-rotary tool bit, describes a helix toolpath by moving more or less linearly while the workpiece rotates. The Turing processes are typically carried out on a lathe, considered to be the oldest of machine tools, and can be of different types such as straight Turing, taper Turing, profiling or external grooving. Those types of Turing processes can produce various shapes of materials such as straight, conical, curved, or grooved workpieces. In general, Turing uses simple single point cutting tools. Each group of workpiece materials has an optimum set of tool angles that have been developed through the years. Nowadays computer can be used to simulate the work of Turing machine. Universal Turing machine can be used to simulate all type of other Turing machines.

References:

1. https://www.youtube.com/watch?v=oCkLiHYQ_t4
2. https://github.com/mCodingLLC/VideosSampleCode