Hannah Thien Truc Nguyen
ECE-2063 - Microprocessors
Professor Hutchins
March 31, 2025

Project 2 - Mighty Mo

Objective

The objective of this project was to code trajectory calculations for the USS Missouri in both C and in MIPS. The main 16-inch guns on the USS Missouri (BB-63) shoot a 2700-pound projectile at an initial velocity of 2500 feet per second and have a maximum range of 24 miles. Given the information above and your knowledge from physics write a program in MARS MIPS that takes a user's input for range (R) in yards and prints out the Time-of-Flight in seconds, Maximum Height reached in feet, and the Angle Trajectory in degrees. The effects of air-resistance and curvature of the earth are neglected.

Introduction

MIPS is a Reduced Instruction Set Computer (RISC) designed for easy instruction pipelining. It is a "load/store" architecture, as all instructions must use register operands. MIPS is one of the many languages to learn and understand how assembly and microprocessors work. It is a simple assembly language to learn and can be used to learn other assembly languages as well. Additionally, learning this will help write more efficient higher-level code. This is the beginning of computer architecture and instruction execution. This report shows and describes the second project of the Microprocessors course.

ENIAC (Electronic Numerical Integrator and Computer) was on of the first general-purpose computers and was developed primarily to calculate artillery firing tables used by Naval ships and Army artillery cannons. The benefit of ENIAC was the increase of performance compared to hand calculations, increasing the speed of calculations from 20 hours to 30 seconds (not including 'programming' time).

C is a high-level object-oriented programming language. It is highly efficient and uses less memory than other languages. It has applications in OS and embedded systems, which is why it is important to learn.

Equipment

1. MIPS Green Sheet
2. MARS MIPS Simulator
3. C compiler
4. Overleaf / LaTeX

Methodology

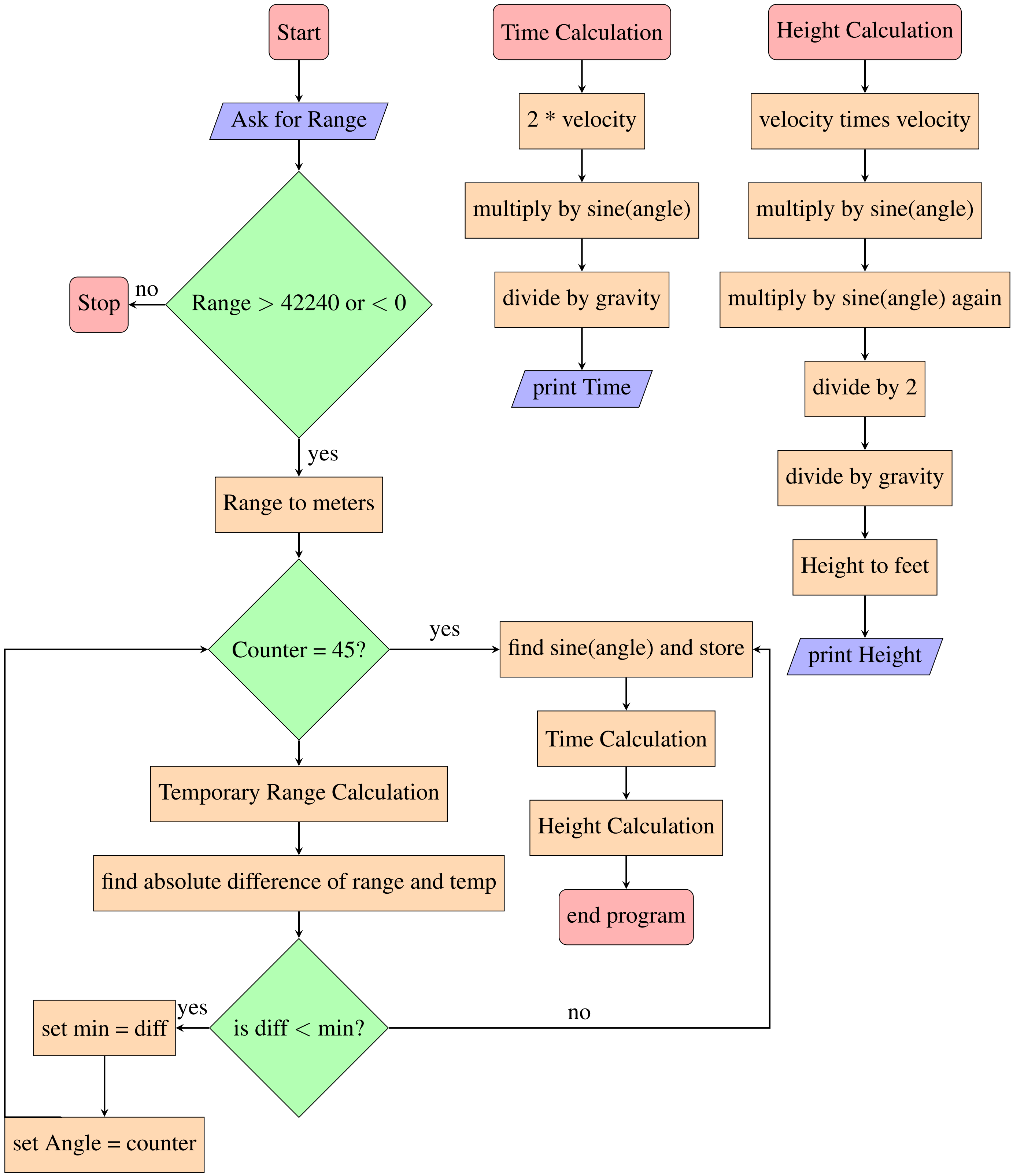
1. Design Requirements
(a) Code the trajectory calculations in C.
(b) Write a program in MIPS that calculates the trajectory
(c) Compare the calculations
2. Description of Design Process

In the first two portions of this project, the goals were to be able to calculate the trajectory of the USS Missouri in both C and in MIPS. The output from these programs will be compared later on. After declaring all variables, values, and asking the user for the range in yards, the first thing that happens is to check that the inputted range is within the guns' actual range of 42240 and 0 yards. If the input is out of range, then the program will just end. If the input is in range, then it will convert the range into meters.

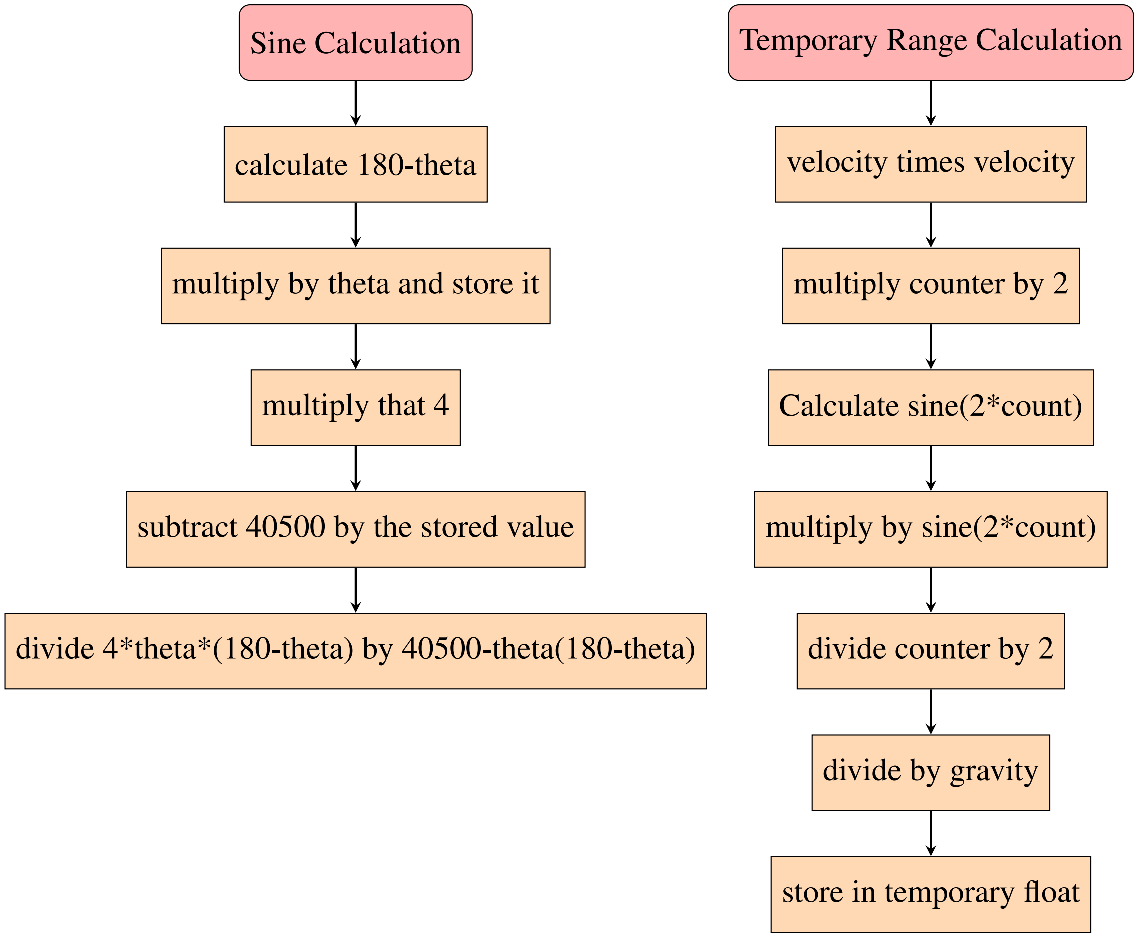
From there, a loop that counts from 0 to 45 degrees will occur. The incremental rate determines the precision of the output. Through each iteration, a temporary range will be calculated using a kinematics equation ($\frac{v^2 sin(2\theta)}{g}$)* to find the absolute difference between the iterated range and the inputted range, where theta is the counter. After finding the absolute difference, it is compared to the current minimum difference. If the absolute difference is smaller, then it will replace the current minimum difference. If not, then the program will continue on, which indicates the angle with the smallest difference and is our angle of trajectory. This method saves time and space since it doesn't have to iterate through all angles.

Note*: any sine calculation is used using the Bhaskara approximation: $\frac{4\times\theta(180-\theta)}{40500-\theta(180-\theta)}$. In the MIPS program, since both the numerator and denominator use $\theta(180-\theta)$, this is first calculated and then multiplied by 4 or subtracted from 40500 for the numerator and denominator, respectively. They are then divided from each other.

Next, since the angle was found, we can use that to find the sine of the angle and store it to use for the next two equations. The first of these equations is time, which is calculated using the equation $\frac{2v sin(\theta)}{g}$. The second of these equations is $\frac{v^2 sin^2(\theta)}{2g}$, which calculates the maximum height of the projectile.



Flowchart of the Process and Calculations



Calculations Flowcharts

Results

As a result of this project, I calculated the trajectory of the 16-inch guns on the USS Missouri in both C and MIPS. After successfully programming the trajectory calculations in both languages, we can compare them. As seen from the figures below, using the same range, the results are nearly identical, confirming that the calculations and code are the same. Theoretically, at the maximum angle of 45 degrees, the range could be farther, but since the effects of drag and the Coriolis effect were neglected, we can assume that 42240 yards is the maximum range.

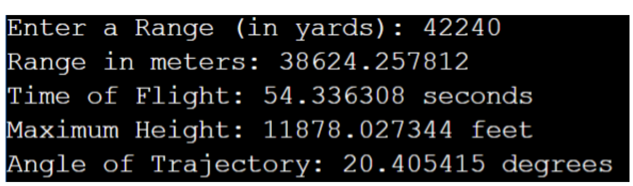


Figure 1: Output from the C program

```
Enter a Range (in yards): 42240

Time of Flight in seconds: 54.336308
Maximum Height in feet: 11878.026
Angle of Trajectory in degrees: 20.405415
-- program is finished running --
```

Figure 2: Output from the MIPS program

Part of this project was to increase the performance compared to hand calculations from 30 hours to 30 seconds, so I tested the limits of each program. I found that the best precision to use for the largest range possible was 0.0003 if a person wanted a calculation that took less than 30 seconds. If the precision was 0.001, which is still very specific, the worst case would take less than 8 seconds to compute. This is specifically for the MIPS version of the program. It seems that the less precise the increment, then time decreases exponentially. For the C version, it takes less than a second to compute no matter the precision. However, there is a limit to the precision. I found that a precision of 0.00002 is the best precision available and anything lower will not compute correctly or have errors. Any errors are due to how the floats are stored and rounded. IEEE 754 isn't completely precise, which leads to marginal errors.

Conclusion

From this project, I successfully created programs in C and in MIPS to calculate the trajectory of a projectile shot by a 16-inch gun on the Mighty Mo. This project helped me learn how to program in C and the differences between that and Java. Additionally, it helped me convert from a higher-level language to a lower-level language. There were some errors while programming MIPS that were harder to debug than in C, such as looping errors since it was harder to keep track of the floats and having to constantly convert from IEEE 754. Overall, this was a fun project and allowed me to program in C and have a good project to put on my resume.

Appendix

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float sine(float theta){
    float degrees = (4*theta*(180-theta))/(40500-theta*(180-theta));
    return degrees;
}

// float cosine(float theta){
//     float degrees = (32400-(4*theta*theta))/(32400+(theta*theta));
//     return degrees;
// }

int main()
{
    float Time, Height, Angle, Range, g = 9.81, v = 762, temp;
    // 24 miles = 38624.256 meters = max range = 42240 yards
    // 2500 ft/s = 762 m/s = initial velocity

    // Ask the user to type a number
    printf("Enter a Range (in yards): \n");
    // Get and save the number the user types
    scanf("%f", &Range);
    // if the inputted range is too large or too little for the cannon to fire,
    if (Range > 42240 || Range < 0){
        printf("Yeah not possible bud"); // then just end the program, ain't no way bro
        exit(0);
    }
    Range *= 0.9144; // get range in meters (metric system >>> imperial)
    printf("Range in meters: %f\n", Range);

    double min_diff = 1e9; // Store the smallest difference
    for (float i = 0; i <= 45; i += 0.001){
        temp = (pow(v, 2)*sine(2*i))/g;

        double diff = fabs(temp - Range);  // Calculate absolute difference
```

Figure 3: C Code part 1

```c
        // printf("Angle: %.4f | temp: %.6f | Range: %.4f |
        // Check if this is the closest value to Range
        if (diff < min_diff) {
            min_diff = diff;
            Angle = i;
        } else {
            break;
        }
    }

    Time = 2*v*sine(Angle)/g;
    Height = pow(v, 2) * pow(sine(Angle), 2) / (2 * g);
    Height *= 3.28084; // convert to feet

    printf("Time of Flight: %f seconds\n", Time);
    printf("Maximum Height: %f feet\n", Height);
    printf("Angle of Trajectory: %f degrees", Angle);
    return 0;
}
```

Figure 4: C Code part 2

```
# The main 16-inch guns on the USS Missouri (BB-63) shoot a 2700-pound projectile at
# an initial velocity of 2500 feet per second and have a maximum range of 24 miles.
# Given the information above and your knowledge from physics write a program in MARS
# MIPS that takes a user's input for range (R) in yards and print out the Time-of-Flight
# (tf light) in seconds, Maximum Height reached (hM AX ) in feet, and the Angle Trajectory (θ0)
# in degrees.
.data
gravity : .float 9.81 # in m/s^2
velocity : .float 762 # in m/s
getRange : .asciiz "Enter a Range (in yards): "
notSlay : .asciiz "BOO NOT POSSIBLE\n"
Time : .asciiz "\nTime of Flight in seconds: "
Height : .asciiz "\nMaximum Height in feet: "
Angle : .asciiz "\nAngle of Trajectory in degrees: "
increment : .float 0.001 # can be changed and decides the precision point
two : .float 2
maxRange : .float 42240
YtoM : .float 0.9144
min_diff : .float 4294967294
maxAngle : .float 45
sineMax : .float 40500
one8ty : .float 180
toFeet : .float 3.28084
zero : .float 0.0

.text
        la, $a0, getRange # get input
        li $v0, 4 # print string
        syscall

        li $v0, 6 # Read float input
        syscall
        mov.s $f20, $f0 # move to another location
```

Figure 5: MIPS Code part 1

```
# $f4 = min_diff, $f5 = Time, $f6 = Height, $f7 = temp, $f8 = Sine(Angle), $f9 = maxAngle
# $f10 = counter, $f11 = diff
# $f16 = theta*(180-theta), $f17 = theta, $f18 = temp4SineFunction, $f20 = Range,
# $f21 = increment, $f22 = gravity, $f23 = velocity, $f24 = maxRange, $f25 = YtoM,
# $f26 = 180, $f27 = 40500, $f28 = meters to feet, $f30 = angle
        lwc1 $f0, zero
        lwc1 $f2, two
        lwc1 $f4, min_diff
        lwc1 $f9, maxAngle
        lwc1 $f21, increment
        lwc1 $f22, gravity
        lwc1 $f23, velocity
        lwc1 $f24, maxRange
        lwc1 $f25, YtoM
        lwc1 $f26, one8ty
        lwc1 $f27, sineMax
        lwc1 $f28, toFeet

# 6 to read, 2 to print
# Checks if range is > 42240 or < 0
c.lt.s $f20, $f0
bc1t exit
c.lt.s $f24 $f20
bc1t exit

mul.s $f20, $f20, $f25 # Range to meters
```

Figure 6: MIPS Code part 2

```
loop:
        c.lt.s $f10, $f9
        bc1f continue
        add.s $f10, $f10, $f21

        mul.s $f7, $f23, $f23 # velcity squared
        mul.s $f17, $f10, $f2
        jal sine
        mul.s $f7, $f7, $f8
        div.s $f7, $f7, $f22  # divide by gravity

        sub.s $f11, $f7, $f20
        abs.s $f11, $f11

        c.lt.s $f11, $f4
        bc1f continue

        mov.s $f4, $f11
        mov.s $f30, $f10

        j loop


sine: # $f17 is theta, save result in $f8 = angle/degrees
        sub.s $f16, $f26, $f17
        mul.s $f16, $f17, $f16

        mul.s $f8, $f16, $f2
        mul.s $f8, $f8, $f2

        sub.s $f18, $f27, $f16
        div.s $f8, $f8, $f18

        jr $ra
```

Figure 7: MIPS Code part 3

```
continue:
        mov.s $f17, $f30
        jal sine
        # Time Calculation
        mul.s $f5, $f2, $f23
        mul.s $f5, $f5, $f8
        div.s $f5, $f5, $f22

        la, $a0, Time
        li $v0, 4
        syscall

        mov.s $f12, $f5
        li $v0, 2
        syscall

        # Height Calculation
        mul.s $f6, $f23, $f23
        mul.s $f6, $f6, $f8
        mul.s $f6, $f6, $f8
        div.s $f6, $f6, $f2
        div.s $f6, $f6, $f22
        mul.s $f6, $f6, $f28 # meters to feet

        la $a0, Height
        li $v0, 4
        syscall

        mov.s $f12, $f6
        li $v0, 2
        syscall

        # Print Angle
        la, $a0, Angle
        li $v0, 4
        syscall
```

Figure 8: MIPS Code part 4

```
        mov.s $f12, $f30
        li $v0, 2
        syscall

exit:
        # safe system call exit
        li $v0, 10
        syscall
```

Figure 9: MIPS Code part 5

Registers | Coproc 1 | Coproc 0

Name	Float	Double
$f0	0x00000000	0x0000000000000000
$f1	0x00000000	
$f2	0x40000000	0x0000000040000000
$f3	0x00000000	
$f4	0x3da00000	0x425958843da00000
$f5	0x42595884	
$f6	0x46399858	0x4716e0a546399858
$f7	0x4716e0a5	
$f8	0x3eb31448	0x423400003eb31448
$f9	0x42340000	
$f10	0x41a33f03	0x3ec8000041a33f03
$f11	0x3ec80000	
$f12	0x41a33e66	0x0000000041a33e66
$f13	0x00000000	
$f14	0x00000000	0x0000000000000000
$f15	0x00000000	
$f16	0x454b899e	0x41a33e66454b899e
$f17	0x41a33e66	
$f18	0x47117b66	0x0000000047117b66
$f19	0x00000000	
$f20	0x4716e041	0x399d49524716e041
$f21	0x399d4952	
$f22	0x411cf5c3	0x443e8000411cf5c3
$f23	0x443e8000	
$f24	0x47250000	0x3f6a161e47250000
$f25	0x3f6a161e	
$f26	0x43340000	0x471e340043340000
$f27	0x471e3400	
$f28	0x4051f948	0x000000004051f948
$f29	0x00000000	
$f30	0x41a33e66	0x0000000041a33e66
$f31	0x00000000	

Figure 10: Floating Point Registers