Q1. (233- 318)

base_converter: 233-298

strlen: 301-318

TYPE 1

First, the string received from the user is sent as an argument to the strlen function called inside the base_converter function. While iterating the string, if the character is not '\0', we continue and increase the value of the register holding the string length value by 1. When the loop ends, the length of the resulting string is returned and stored in the $s2 register. After checking whether it is 1 or 0 without using a loop for msb, if it is 1, (-)2^(strlen - i - 1) is performed and the result is added to $s0. Since there is no need to calculate two's complement if 0, the result is added to $s by continuing the loop and if the character is 1, 2^(strlen - i - 1) is performed in each iteration.

Sample run:

```
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 1
input: 1101
type: 1
-3
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 1
input: 0111010
type: 1
58
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 1
input: 100000
type: 1
-32
```

Q2. (323-378)

add_rational_numbers: 323-378

After receiving a parameter from the user, the add rational numbers function is called and the mathematical operations given in the question are translated into mips code. After the max(numerator, denominator) operation, the largest value is assigned to $t0 and the greatest common divide algorithm is used. With the value returned by gcd, the numerator and denominator are divided and the simplest form is achieved. The python version of the process I simply did in mips

```
while b>0:
 r=a%b
 a=b
 b=r
```

Sample run:

```
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 2
Enter the first numerator: 3
Enter the first denominator: 6
Enter the second numerator: 2
Enter the second numerator: 10
3/6+2/10=7/10
Main Menu:
```

Q3. (639-694)

text_parser: 639-694

By creating a nested loop, we check if each character in the input array matches the characters in the parser array, and if there is no match in the inner loop, we print the character else we move to a new line.

Sample run:

```
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 3
Input text : you just-wait sunshine. you just_wait
Parser characters: -_
you
just
wait
sunshine.
you
just
wait

Main Menu:
```

Q4. (385-632)

parsed_string: 385-470

power: 432-444

matrix_operations: 473- 593

sqrt: 597-633

First of all, parse_string($a0= &userInput, $a1 = &int_array) function iterates the characters of the string, when it sees whitespace, it separates the string and jumps to the str_to_int label. The split string is converted to integer here and saved to int_array[]. Since we couldn't find a way to do the conversion in one go, we iterated the split string to integer the character in each iteration,

```
lb $t1, 0($t0)
andi    $t4 $t1, 0x0F           # Mask the first four bits
```

and multiplied 10^(strlen - i - 1) by parsed_string[i] to add $t3.

ex. : '105' -> 10^(3 - 0 - 1) * 1 + 10^(3 - 1 - 1) * 0 + 10^ (3 - 2 - 1)* 5 = 105

After the int_array[] is completed, the function returns the length of the array and

before proceeding with the matrix operations, the height and width of the matrix are determined by taking the square root of the length. (since it's a square matrix, both are the same). In order for the algorithm to be more understandable, I wrote the same algorithm in python.

```
matrix =[[3,8,12,1,2,3],[4,3,2,4,5,4], [5,6,7,11,4,2],
         [9,3,23,14,5,58], [5,3,4,5,7,8], [4,2,4,9,1,2]]
M = 6
N = 6
horizontal_array= []
vertical_array = []
for i in range(0,M,2): ### horizontal part
    temp_result = 1
    for y in range(N):
        if y % 2 == 1:
            update_i = i +1
        else:
            update_i = i
        temp_result *=matrix[update_i][y]
    horizontal_array.append(temp_result)
for i in range(1,N,2): ## vertical part
    temp_result = 1
    for y in range(M):
        if y % 2 == 1:
            update_i = i -1
        else:
            update_i = i
        temp_result *=matrix[y][update_i]
    vertical_array.append(temp_result)
print(horizontal_array)
print(vertical_array)
```

In mips code, unlike Python, you can enter a squared matrix of any size as long as n is even, and we print the results directly instead of keeping them in a separate array.

Sample run:

```
Main Menu:
1. Base Converter
2. Add Rational Number
3. Text Parser
4. Mystery Matrix Operation
5. Exit
Please select an option: 4
input: 6 4 8 9 5 4 7 5 12 1 15 13 14 15 18 16
960     43200
280     14742
```