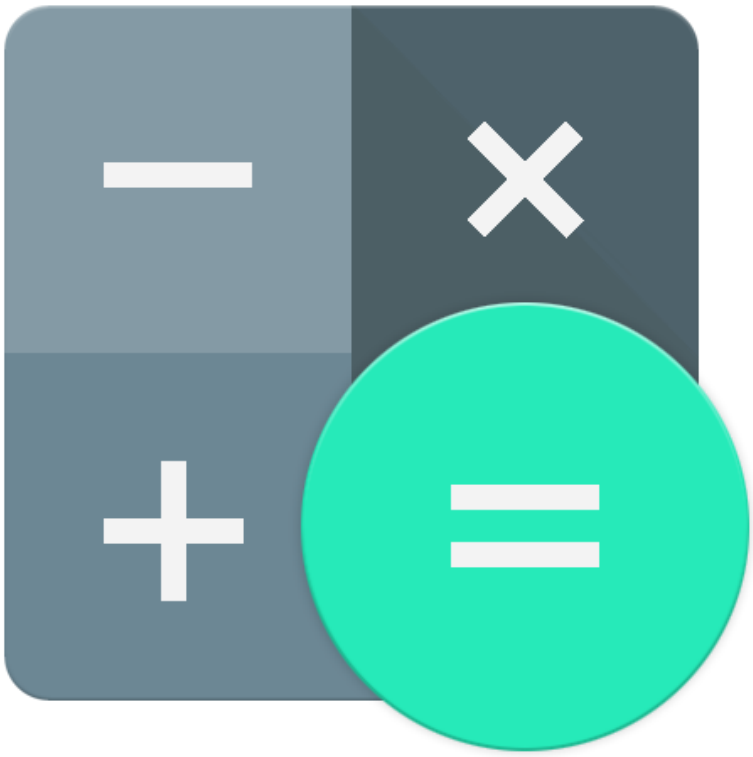
DOCUMENTACION DE LA CALCULADORA EN BINARIO



AUTOR: DIEGO JESUS MUÑOZ ANDRADE

GRUPO: 1-C BIS

MATRICULA: 23640125

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PIEDAD

PROBLEMA:

Realizar una calculadora que pueda sumar, restar, multiplicar o dividir en cualquiera de los tres sistemas numéricos (binario, octal, hexadecimal).

DATOS DEL PROGRAMA

Nombre: calculadora en binario.
Sistema numérico: binario.
Herramientas: Raptor Avalonia, Python.
Creador: Diego Jesús Muñoz Andrade.
Institución: Tecnológico Nacional de México.
Plantel: Campus La Piedad

ANALISIS

INICIO:

Empezaremos definiendo 6 variables (state, bits1, bits2, sig, xkey, String_) y empezaremos dibujando la interfaz de usuario.

ENTRADA:

Mediante un ciclo, el programa estará atento para verificar que se esté ingresando algún dato con el teclado y posterior evaluaremos el dato. Evaluaremos si se ingreso un 0 o 1 para empezar a guardarlo en la variable bits1, si se presiona un signo (+, -, *, /) entonces guardamos el signo para realizar la operación y ahora guardamos los 0 y 1 que el usuario ingrese en la variable bits2. Si se presiona la tecla Enter, continuamos con el proceso.

PROCESO:

Para que esta calculadora pueda operar, necesitamos convertir los datos en binario a décima, para eso se necesita definir 4 variables (num1, num2, result, j, mult) donde "j <- lenght_of(bits1)". Mediante un ciclo verificamos si "j < 1", de no ser así verificamos si "bits[j]" contiene 0 o 1, si contiene 1, entonces a resultado le agregamos el valor de "mult", cuyo valor se va multiplicando por 2 en cada corrida del ciclo. Este es el mismo proceso que se utiliza en la conversión de la segunda cadena de bits.

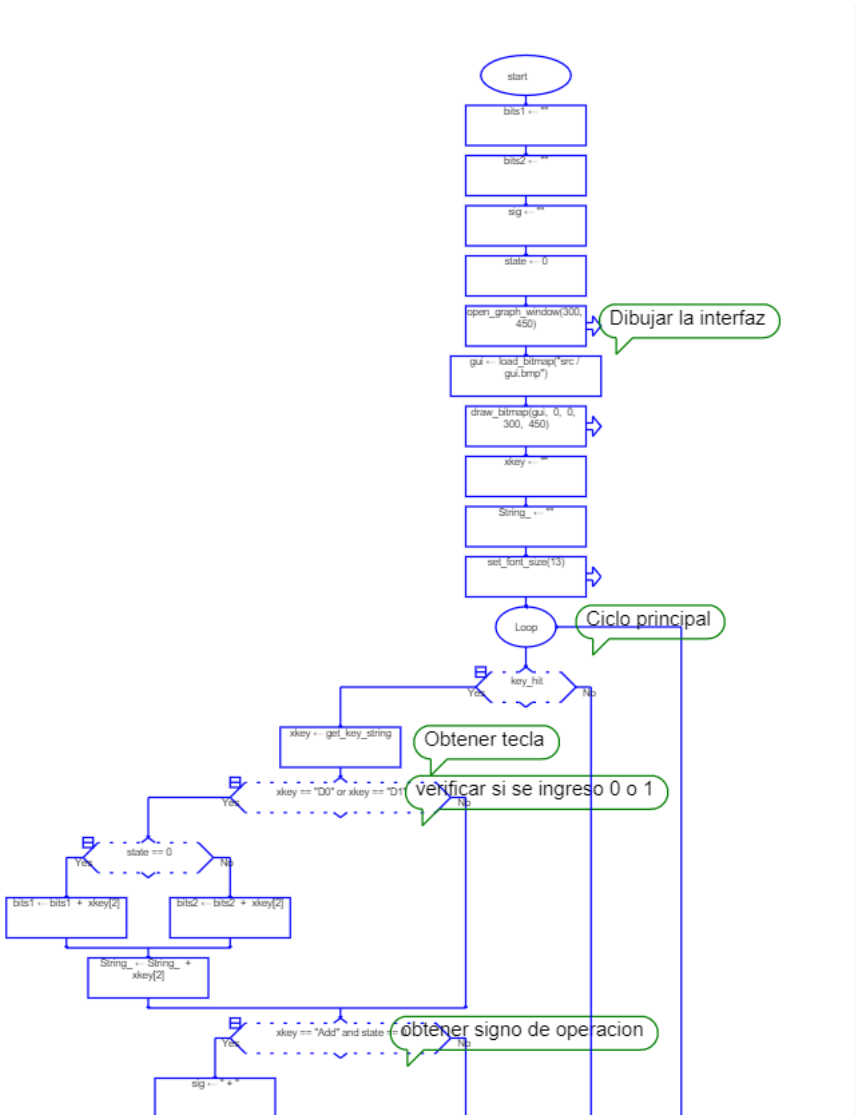
Para realizar la operación elegida por el usuario evaluaremos mediante 4 condiciones la variable "sig" y verificaremos el símbolo guardado, si "sig es igual a +" haremos una suma, si "sig es igual a –" entonces haremos una resta, si "sig es igua a *" entonces haremos una multiplicación y si no entonces será una división, el resultado de dicha operación se guardara en la variable "result".

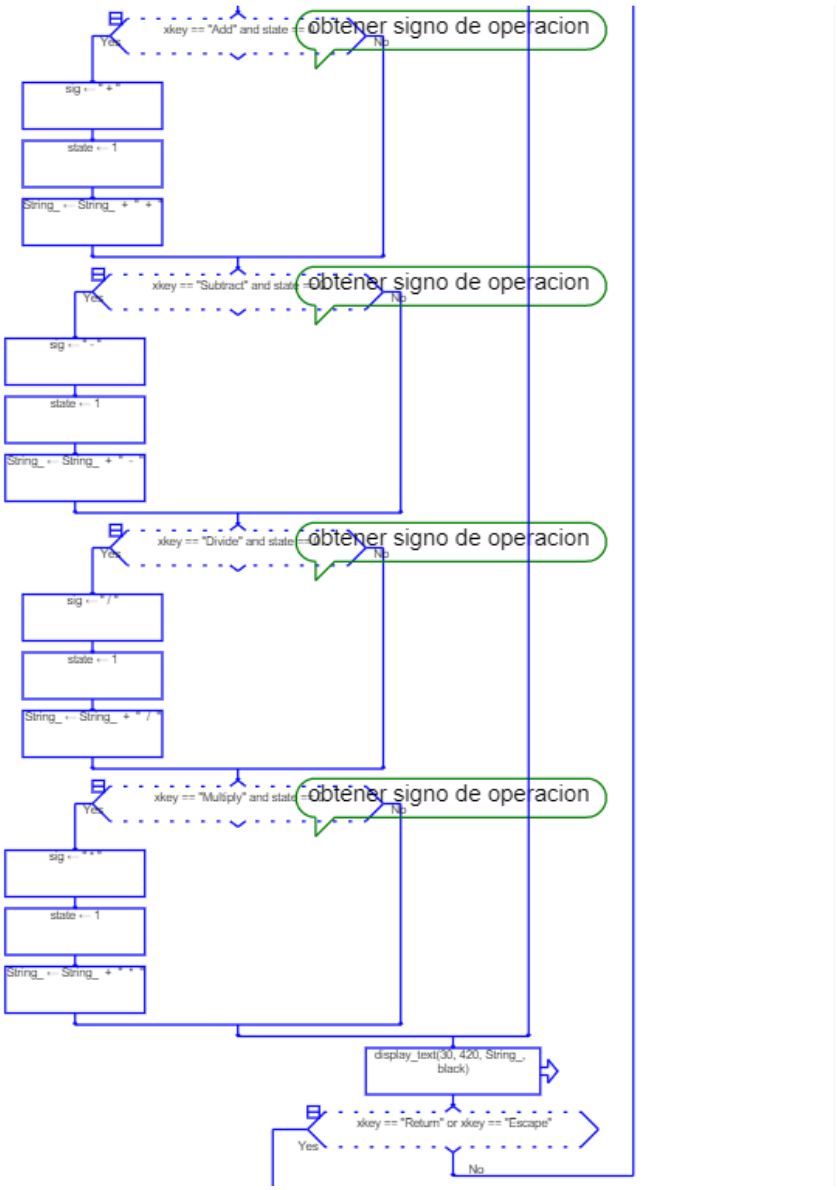
Ahora es necesario convertir el resultado a binario, ya que este esta en decimal. Se declaran 4 variables (bits_al_reves, división <- result, residuo, i). Para la conversión se usara un ciclo donde la condición es “división < 2”, dentro del ciclo obtendremos el residuo entre 2 y lo guardaremos en residuo y dividiremos /2 y guardamos el valor en división. El residuo será guardado en la variable “bits_al_reves” e incrementamos i de 1 en 1.

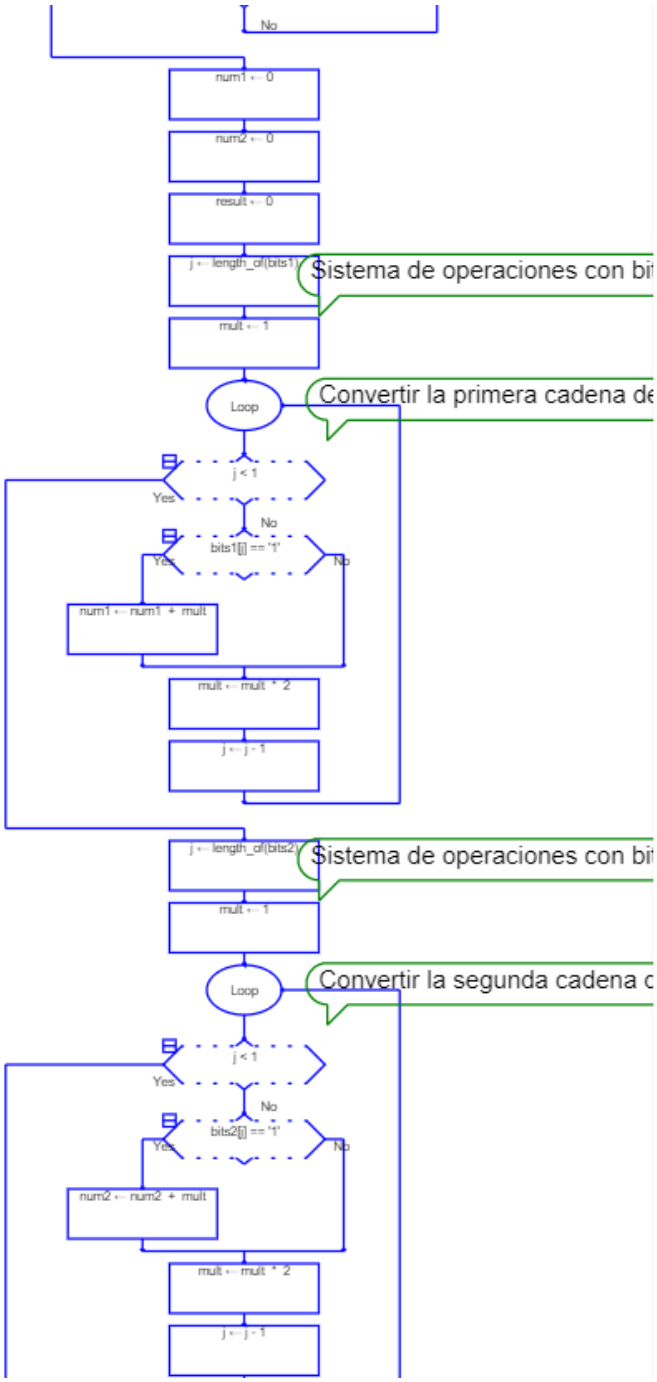
SALIDA:

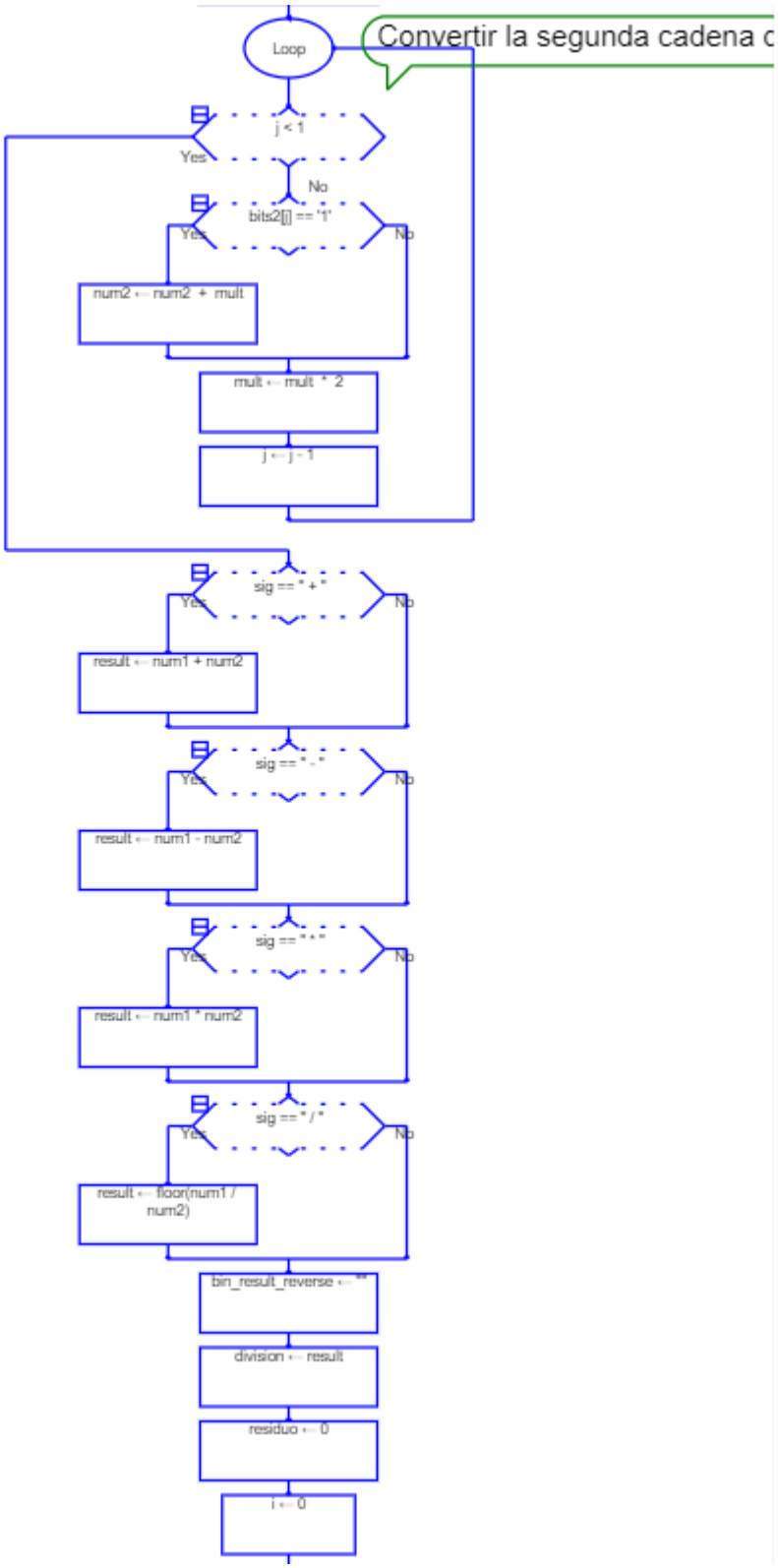
Para poder imprimir los bits correctamente (ya que están al revés), esto se hará mediante un ciclo donde “i < 1”. De no ser así mostraremos la posición de la variable “bits_al_reves[i]” y decrementamos i de 1 en 1.

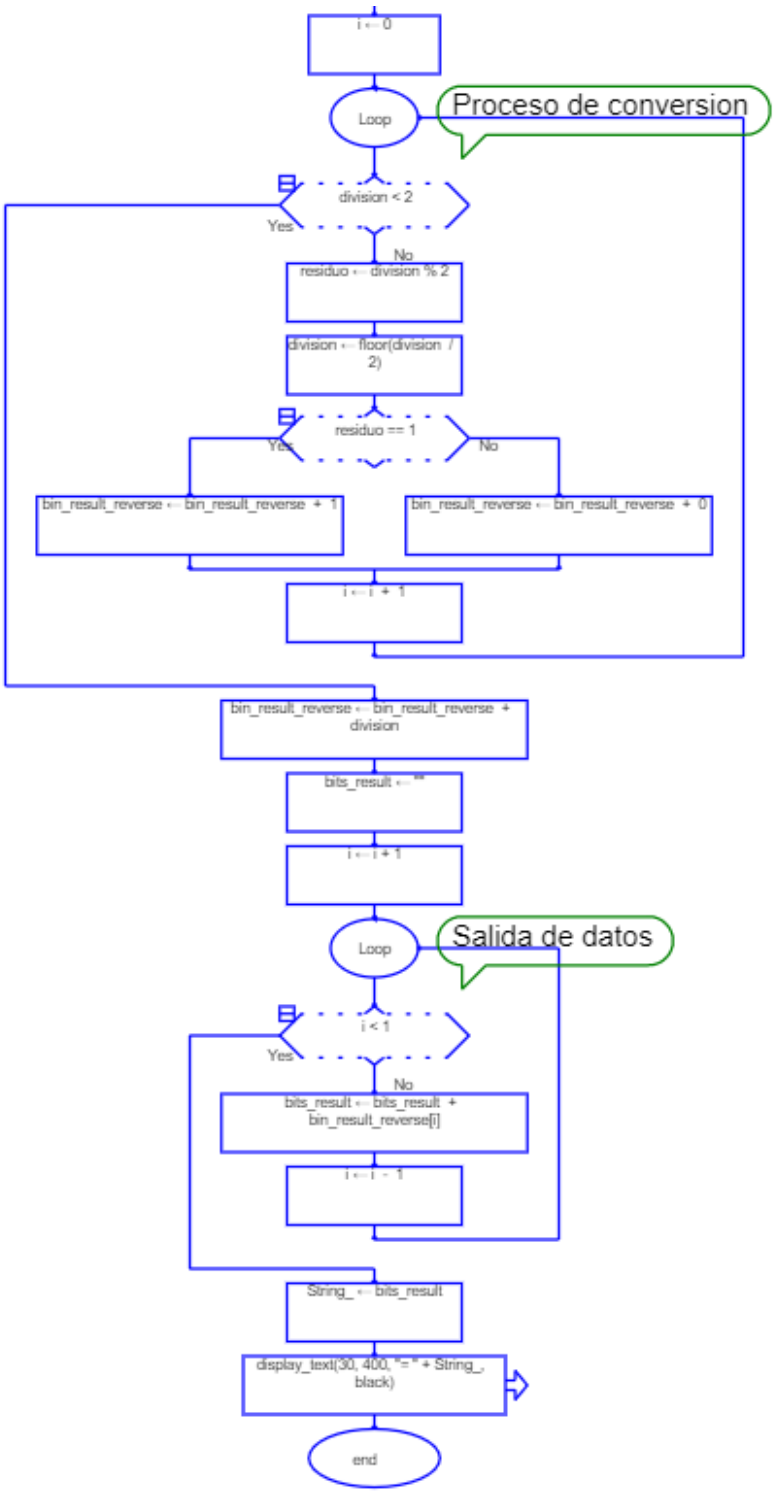
DIAGRAMA DE FLUJO




obtener signo de operacion
xkey == "Add" and state
Yes
No
sig ← " + "
state ← 1
String_ ← String_ + " + "
obtener signo de operacion
xkey == "Subtract" and state
Yes
No
sig ← " - "
state ← 1
String_ ← String_ + " - "
obtener signo de operacion
xkey == "Divide" and state
Yes
No
sig ← " / "
state ← 1
String_ ← String_ + " / "
obtener signo de operacion
xkey == "Multiply" and state
Yes
No
sig ← " * "
state ← 1
String_ ← String_ + " * "
display_text(30, 420, String_, black)
xkey == "Return" or xkey == "Escape"
Yes
No


No
num1 ← 0
num2 ← 0
result ← 0
j ← length_of(bits1)
Sistema de operaciones con bi
mult ← 1
Loop
Convertir la primera cadena de
j < 1
Yes
No
bits1[j] == '1'
Yes
No
num1 ← num1 + mult
mult ← mult * 2
j ← j - 1
j ← length_of(bits2)
Sistema de operaciones con bi
mult ← 1
Loop
Convertir la segunda cadena d
j < 1
Yes
No
bits2[j] == '1'
Yes
No
num2 ← num2 + mult
mult ← mult * 2
j ← j - 1


Loop
Convertir la segunda cadena c
j < 1
Yes
No
bits2[j] == '1'
Yes
No
num2 <- num2 + mult
mult <- mult * 2
j <- j - 1
sig == "+"
Yes
No
result <- num1 + num2
sig == "-"
Yes
No
result <- num1 - num2
sig == "*"
Yes
No
result <- num1 * num2
sig == "/"
Yes
No
result <- floor(num1 / num2)
bin_result_reverse <- ""
division <- result
residuo <- 0
i <- 0


i ← 0
Loop
Proceso de conversion
division < 2
Yes
No
residuo ← division % 2
division ← floor(division / 2)
residuo == 1
Yes
No
bin_result_reverse ← bin_result_reverse + 1
bin_result_reverse ← bin_result_reverse + 0
i ← i + 1
bin_result_reverse ← bin_result_reverse + division
bits_result ← ""
i ← i + 1
Loop
Salida de datos
i < 1
Yes
No
bits_result ← bits_result + bin_result_reverse[i]
i ← i - 1
String_ ← bits_result
display_text(30, 400, "= " + String_, black)
end