Universidad de Puerto Rico – Mayagüez
Departamento de Ingeniería en Ciencias en Computación
CIIC 4082 - Arquitectura de Computadoras II

Calculadora con Launchpad MSP430FR6989

Prof. Jose Navarro-Figueroa
Grupo 18: Jomar Santos, Ignacio Tampe,
Peter L. Santana, Dianelys Saldana
Martes, 16 de marzo de 2021

Índice

Introducción

Este reporte tiene como objetivo demostrar en detalle el funcionamiento general de una calculadora implementada con el *Launchpad MSP430FR6989* como herramienta principal. En este además se estarán explicando cada una de las funcionalidades y subrutinas que componen la calculadora, de manera que se pueda entender a modo general los procesos detrás del funcionamiento explícito de la misma.

La implementación de la calculadora fue realizada de manera que esta tenga la capacidad de llevar a cabo operaciones aritméticas básicas como lo son la suma, resta, multiplicación y división. Como parte de la implementación, se diseñó una interfaz la cual incluye 6 dígitos, un símbolo negativo (on/off) y dos botones para apretar (push buttons). Algunos de los requisitos para la implementación de la calculadora son los siguientes:

- Todos los operandos deben tener 3 dígitos.
- La adición debe producir 3 o 4 dígitos como resultado.
- La resta debe producir un resultado de 3 dígitos.
- Los productos deben tener 6 dígitos.
- Los cocientes deben tener 3 dígitos.
- En el caso de la división, no se mostrará el residuo.
- Se incluirán los “leading 0 's” según sean necesarios.
- Para la adición se utilizaran 4 dígitos de no haber “leading 0 ’s”.
    - En los demás casos se utilizarán 3 dígitos.
- Cada click en el botón izquierdo incrementará el dígito por 1.
- Cada click en el botón derecho hace el próximo dígito activo.
- Luego de ingresar los 3 dígitos, al presionar el botón derecho se activará la opción de operadores.
    - Luego, al presionar el botón izquierdo nuevamente se podrá elegir entre los operadores disponibles.
    - Al presionar el botón derecho se enviará confirmación del operador y se podrá comenzar a escribir el próximo operando.
- Luego de entrar los 3 dígitos, al presionar el botón derecho se mostrará el resultado.
    - En el caso de que el resultado sea negativo, se mostrará el signo correspondiente. De no ser negativo, este se mantendrá oculto.

Proceso de solución del problema

El problema que se nos presentó fue el de realizar una calculadora en “Assembly Language” para el Launchpad MSP430FR6989. Nuestra interfaz disponible incluye 6 dígitos disponibles en la pantalla con un signo negativo y dos botones presionables. Se nos pidió que todos los operandos tuvieran 3 dígitos. Adicionalmente, otro de los requerimientos fue que el número incrementará con el botón izquierdo y luego de esto aparecieran las posibles operaciones en pantalla siendo suma, resta, multiplicación y división.

Para empezar, decidimos realizar una tabla con distribución de las tareas necesarias para la implementación de la calculadora. Se dividieron las rutinas en 4 categorías: Graphics, Arithmetic, Input y Main. Decidimos realizar un sistema de dificultad preliminar para poder distribuir las tareas equitativamente; 3 representando difícil, 2 siendo moderado y 1 fácil. Buscamos que en promedio cada integrante tuviera 4 puntos en dificultad entre las tareas divididas, aunque algunas luego resultaron ser más fáciles o difíciles de realizar de lo que se estimó inicialmente.

Luego de dividir las tareas comenzamos a buscar como otras personas habían implementado sus calculadoras en sistemas similares al launchpad o que por lo menos hubieran utilizado assembly para crear la calculadora. Encontramos varios videos que, a pesar de no utilizar un launchpad para implementarlos, nos dieron una idea de cómo se podrían implementar las subrutinas en nuestro sistema. [1] Al terminar nuestra búsqueda, encontramos que muchos de los conceptos que necesitábamos para poder implementar la calculadora ya habían sido discutidos en clase, así que repasamos las lecciones aprendidas antes de decidir cómo implementar nuestras subrutinas.

Distribución de tareas

La siguiente tabla muestra un desglose de las tareas (investigaciones, pruebas, algoritmos, creación de código, etc.), así como los integrantes del equipo que realizaron las mismas.

Graphics	DrawInt	Takes an Int from R5 and draws it on the LCD starting from a position on R6. Clears the LCD at the required positions before drawing	Ignacio Tampe	3
Graphics	DrawSign	Takes the value on R7 and draws a sign (+, -, *, or /) (0, 1, 2, or 3) onto the LCD at the position on R6. Clears the LCD character at position R6 before drawing	Ignacio Tampe	2
Arithmetic	Add	Adds R8 and R9 together. Saves result to R10	Peter Santana	1
Arithmetic	Subtract	Subtracts R9 from R8. Saves result to R10	Peter Santana	1
Arithmetic	Multiply	Multiplies R8 and R9. Saves result to R10	Dianelys Saldana	1
Arithmetic	Divide	Divides R8 by R9. Saves result to R10	Peter Santana	2
Arithmetic	Operate	Subroutine that branches off to the other arithmetic operations based on the specified sign in R7	Dianelys Saldana	1
Input	EnterInt	Prompts the user for an int. Saves the result to R9	Jomar Santos	3
Input	EnterSign	Prompts the user for an operation (+, -, *, or /) (0, 1, 2, or 3). Saves the result to R7	Dianelys Saldana	2
Main	Main Routine	Main routine that: ● Runs EnterInt ● Moves R9 to R8 ● Runs EnterSign ● Runs EnterInt ● Runs Operate ● Clears the LCD ● Draws the result	Jomar Santos	3

3 - “Difícil”, 2 - “Moderado”, 1 - “Fácil”

17 Puntos de dificultad en total

LINK DEL VIDEO:

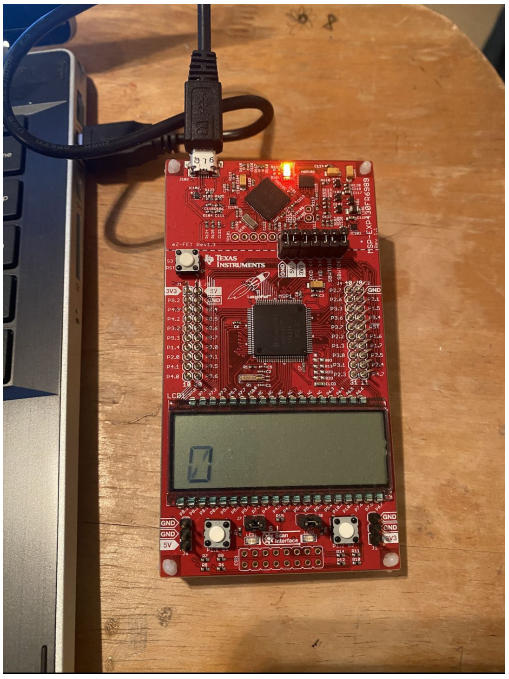
(205) Calculator - Group 18 - YouTube

Imágenes del producto

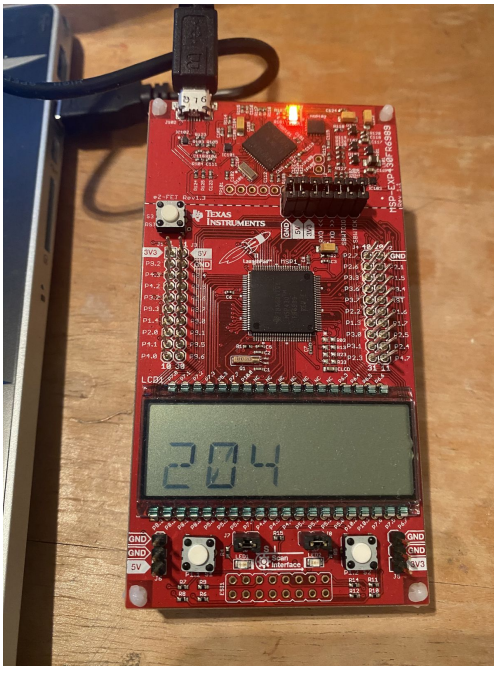
Las siguientes imágenes muestran el funcionamiento de varias operaciones aritméticas utilizando el MSP430FR6989.

- **Ejemplo de operación de suma (204 + 62 = 266):**

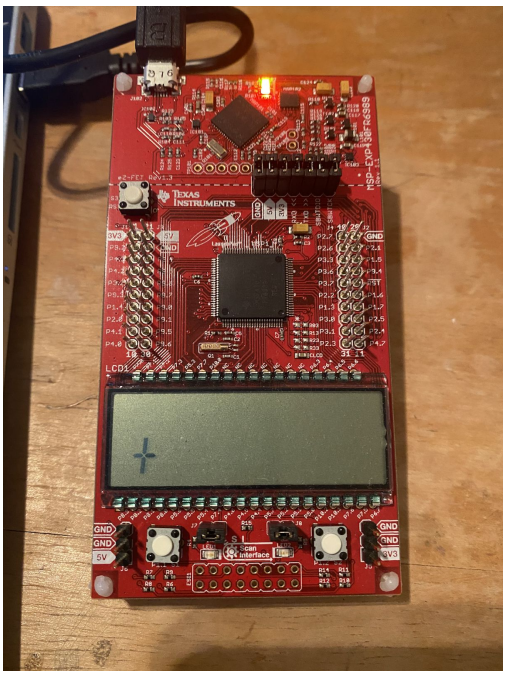
1. Estado inicial del sistema (0)



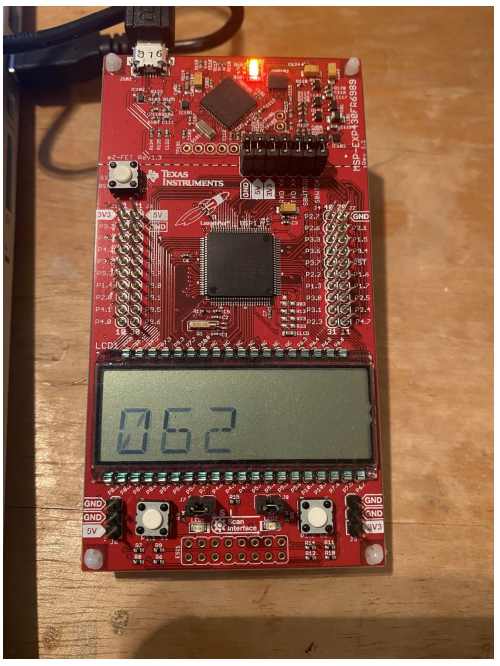
2. Imagen con operando inicial (204)



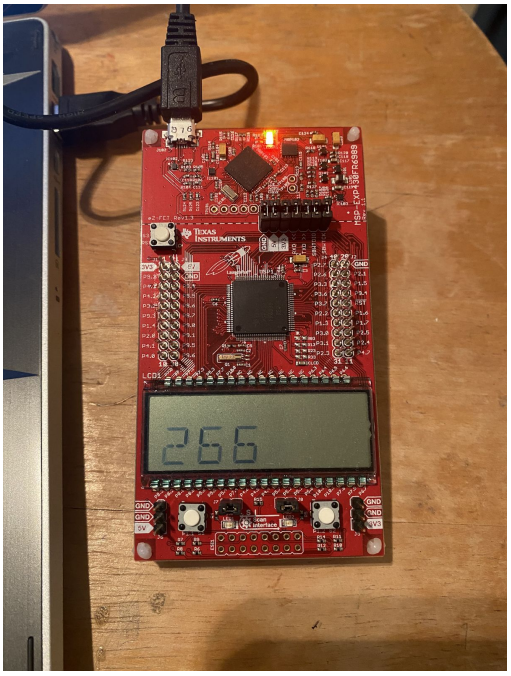
3. Imagen con operador (+)



4. Imagen con el segundo operando (62)

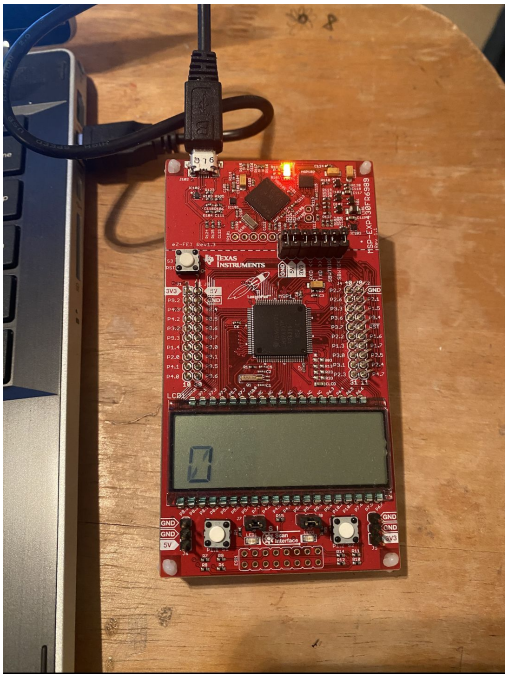


5. Imagen con el resultado (266)

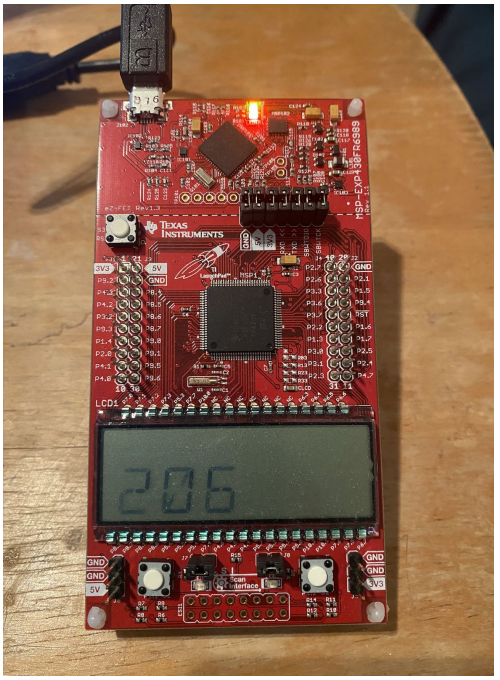


- **Ejemplo de operación de multiplicación (206 * 54 = 11,124):**

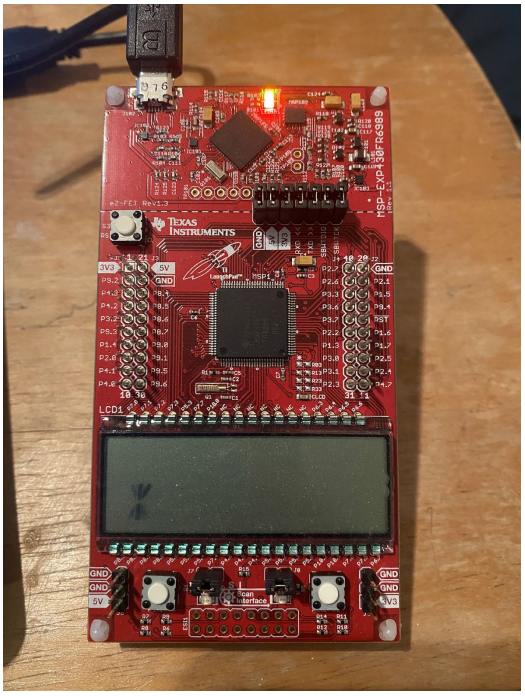
1. Estado inicial del sistema (0)



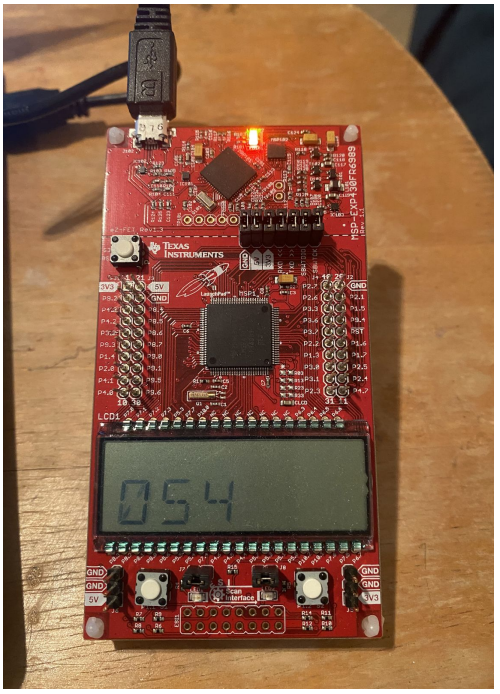
2. Imagen con operando inicial (206)



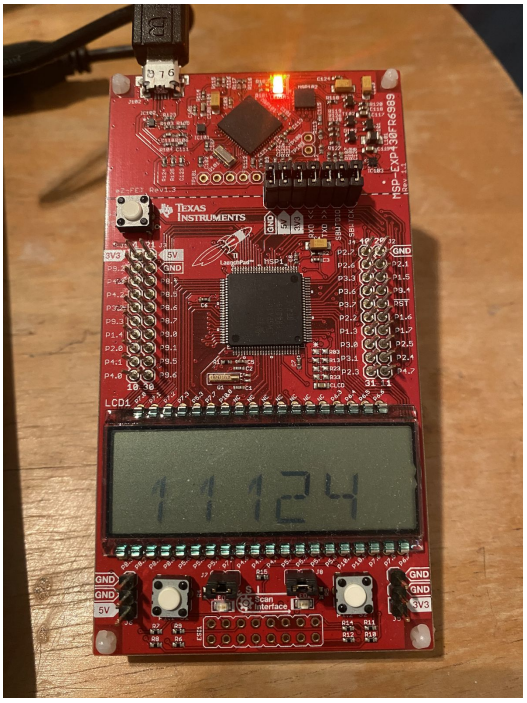
3. Imagen con operador (*)



4. Imagen con el segundo operando (54)



5. Imagen con el resultado (11,124)



Código del programa

```
;MSP LCD Demo

#include "msp430.h"                     ; #define controlled
;include file

        NAME    main                    ; module name

        PUBLIC  main                    ; make the main label
                                        ; visible
                                        ; outside this module

         ORG     0FFFEh
         DC16    init                   ; set reset vector to
                                        ; 'init'  label

         ORG     01C00h                 ; Start of RAM

;Digits      0    1     2     3     4     5     6     7     8
DigitH   db 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0,

;Digits (continuation)
  8     9     +     -     *     /
0xFF, 0xE7, 0x03, 0x03, 0x00, 0x00

DigitL   db 0x28, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x50, 0x00, 0xFA, 0x28

LCDADD   db     0xa29, 0xa25, 0xa23, 0xA32, 0xA2E, 0xA27
;LCDADDL db     0xa2a, 0xa26, 0xa24, 0xA33, 0xA2F, 0xA28

         RSEG    CSTACK                 ; pre-declaration of
                                        ; segment
         RSEG    CODE                   ; place program in
                                        ; 'CODE' segment

init:    MOV     #SFE(CSTACK), SP       ; set up stack

main:    NOP                            ; main program
         MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer

       ;Enable LCD Segments 0-21; 26-43
```

```
         MOV.W   #0xffff,&LCDCPCTL0
         MOV.W   #0xfc3f,&LCDCPCTL1
         MOV.W   #0x0fff,&LCDCPCTL2


SetupP1:
        bic.b   #0xFF,&P1SEL0          ; Set PxSel0 and PxSel1
                                       ; PxSell to digital I/O
        bic.b   #0xFF,&P1SEL1          ; Digital I/O is the
                                       ; default
        bic.b   #0xFF,&P9SEL0
        bic.b   #0xFF,&P9SEL1

        mov.b   #11111001B,&P1DIR       ; Set P1.1 and P1.2
                                        ; For input  and all
                                        ; other P1 pins for
                                        ; output
        bis.b   #0xFF,&P9DIR            ; Set all P9 pins for
                                        ; output

        mov.b   #00000110B,&P1REN       ; Activate P1.1 and
                                        ; P1.2 programmable
                                        ; pull-up/pull-down
                                        ; resistors
        bis.b   #00000110B,&P1OUT       ; Set resistors for
                                        ; P1.1 and P1.2
                                        ; as pull-down
        bic.b   #0x01,&P1OUT            ; Clear P1.0 and P9.7
                                        ; output latch to
        bic.b   #0x80,&P9OUT            ; start with both off

UnlockGPIO:                             ; Disable the GPIO
                                        ; power-on default
        bic.w #LOCKLPM5, &PM5CTL0       ; High-impedance mode
                                        ; to activate
                                        ; previously
configured port settings
        ;Initialize LCD_C
        MOV.W   #0x041e,&LCDCCTL0

        ;VLCD generated internally,
        ;V2-V4 Generated internally, v5 to ground.
        ;set VLCD Voltage to 2.6V
```

```
        ;Enable Charge pump and select internal reference for
        ;it.

        MOV.W   #0x0208,&LCDCVCTL

        MOV.W   #0x8000,&LCDCCPCTL     ;Clock sync Enabled

        call #ClearLCD                 ;Clear LCD Memory

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        MOV.B #1,R6 ;Points to first digit spot.
        MOV.W #0,R5
        mov.b #0,R7

nextDigitSetup:
        ;Setting up stuff for nextDigit
        mov.b   #9,R14
        mov.b   #0,R9
        mov.b   #0,R15
        mov     #0,R10

 Mainloop:                             ;------Jomar Santos------
        Call    #start                        ;Starts calculator
;which when it finishes an operation and right clicked when
;displaying result, it loops back to Mainloop
        jmp     TheEnd                        ;Ends program in
                                              ;case of edge case


start:
        Call    #ClearReg                     ;Clears registers
;to start gathering data and setting the fields for the
;calculator process

        Call    #StartDisplay                 ;Displays the first
;segment with a 0 to start the enterInt process
        Call    #EnterInt                     ;Receives input
;pertaining the first integer for the operation
        mov     R9,R8                         ;Since the input
;integer is saved in R9, here we move it to R8 which is the
;register that will hold the first integer for the operation.
```

```
        Call    #nextSign                   ;Starts the process
;for picking which operation will be carried out. (+,-,*,/)
        Call    #Delay                      ;Gives some time
;for the data to be processed correctly

        Call    #StartDisplay               ;Displays the first
;segment with a 0 to start the enterInt process
        mov     #0,R10                      ;Clears R10, since
;this is the register supposed to hold the result values. This
;is done in case of an edge case. This way we avoid issues
;that may occur
        mov.b   #0,R15                      ;Clears R15 back to
;0 to keep track of inputs
        Call    #EnterInt                   ;Receives input
;pertaining the second integer for the operation

        Call    #Operate                    ;Carries out the
;operation chosen with the use of the 2 integers that were
;provided. The result of the operation is saved on R10. R8->
;first integer, R9-> second integer, R10-> Result.

        Call    #ClearLCD                   ;Clears the display
;to prepare for displaying the result for the operation.
        mov.b   #0,R15                      ;Clears R15 back to
;0 to keep track of inputs.
        mov     R10,R5                      ;Moves result to
;R5. The program displays the values inside R5 when the draw
;functions are called or jumped to.
        Call    #DrawResult                 ;Draws the result
;of the operations on the display
        Call    #isNegative                 ;If the result was
;negative, this will add the negative sign to the display.

        mov     #0,R10                      ;Resets R10 ->
;results back to 0, in preparation for the next run of the
;calculator.
        Call    #HoldResult                 ;Displays the
;result until the launchpad is right clicked to start the
;process again.
        jmp     Mainloop                    ;When right clicked
the program will jmp to Mainloop and carry out the program
again
```

DrawResult: ;------Jomar Santos------

```
        Mov     #1,R6                       ;Moves 1 to R6
;which signals the segment in which we will be writing.
        Mov     #0,R8                       ;R8 will be used as
;a counter and for said reason we move a 0 to it.
        cmp     #0,R12                      ;R12 holds half of
;the result for multiplications when this exceeds 1000. This
;is done to be able to display results up to 999,999.
        jeq     DrawInt                     ;If equal then the
;drawing on the display will be carried out normally as done
;with division, subtraction and sum.

        Call    #Drawing100                 ;If the previous
;condition is not met, we start breaking down the result into
;100, 10 and 1, displaying first the half stored in R12.
        Call    #Drawing10
        Call    #Drawing1

        Mov     R10,R12                     ;We move the other
;half of the result in R10, to R12, to carry out the previous
;process and display the other half.
        Call    #Drawing100
        Call    #Drawing10
        Call    #Drawing1

        jmp     fin


Drawing100:                         ;------Jomar Santos------
        cmp     #2,R6                       ;Compare in which
;segment we are drawing at the moment.
        jge     DrawSecSeg100               ;If we are not on
;the 1st segment (R6 >= 2) then we jmp to DrawSecSeg100, to
;avoid the program from not writing 0's when they need to be
;drawn.

        cmp     #100,R12                    ;Compare that R12 is
;greater than 100.
        jge     reduce100                   ;If R12 is greater
;than 100, we proceed to writing the centesimal position on
;the register.

        jmp     fin                         ;If no condition
;were met we jump to the next number in the integer. In this
;case we would jump to the decimals, or multiples of 10.
```

```
DrawSecSeg100:                          ;------Jomar Santos------
        cmp     #100,R12
        jge     reduce100                   ;Compare if 12 is
;greater or equal to 100. If so, we reduce 100 from it and
;increase R8 by 1 in the reduce100 function.

        Mov     R8,R5                       ;If the condition is
;not met then we move R8 to R5, to display in DrawIntDirect.
        Call    #DrawIntDirect

        Inc     R6                          ;We increase R6 by
;one to point to the next segment in which we will be writing.
        Mov     #0,R8                       ;We clear R8, since
;this is the counter. This way we can use it for the next
;positions.
        jmp     fin

Drawing10:                              ;------Jomar Santos------
        cmp     #10,R12                 ;Same process as with
;the Drawing100 function but with 10 instead.
        jge     reduce10
        cmp     #2,R6
        jge     Draw10
        jmp     fin

Drawing1:                               ;------Jomar Santos------
        cmp     #1,R12                  ;Same process as with
;the Drawing100 function but with 1 instead.
        jge     reduce1
        cmp     #2,R6
        jge     Draw1
        jmp     fin

reduce100:                              ;------Jomar Santos------
        Inc     R8                      ;Since the condition
;to enter the function was met we, increase R8 bby one. This
;will keep track of the number of times we reduce the number
;by 100 and is the number which will be displayed.
        sub     #100,R12                ;Subtract 100 from
;the half of the result stored in R12.

        cmp     #100,R12
```

```
        jlo     Draw100                 ;If R12 is lower than
;100 we proceed to draw the number stored in R8. Example if we
;had 900, we subtract 100, 9 times and this is stored in R8
;which is what we want to display.
        jmp     reduce100               ;If the number is not
;lower than 100, then we keep reducing until it is.

Draw100:                                ;------Jomar Santos------
        Mov     R8,R5                   ;We move R8 to R5
;since we display values inside R5.
        Call    #DrawIntDirect          ;We write what's
;stored in R5 to the display

        Inc     R6                      ;We increase R6 by
;one since this points to the segment in which we want to
;write on next.
        Mov     #0,R8                   ;We reset R8 (our
;counter) by passing it a 0, to prepare use in the next stage.
        jmp     fin

reduce10:                               ;------Jomar Santos------
        Inc     R8                      ;Same process as with
;the reduce100 function but with 10 instead.
        sub     #10,R12
        cmp     #10,R12
        jlo     Draw10
        jmp     reduce10

Draw10:                                 ;------Jomar Santos------
        Mov     R8,R5                   ;Same process as with
;the Draw100 function but with 10 instead.
        Call    #DrawIntDirect
        Inc     R6
        Mov     #0,R8
        jmp     fin

reduce1:                                ;------Jomar Santos------
        Inc     R8                      ;Same process as with
;the reduce100 function but with 1 instead.
        sub     #1,R12
        cmp     #1,R12
        jlo     Draw1
        jmp     reduce1
```

```
Draw1:                                  ;------Jomar Santos------
        Mov     R8,R5                       ;Same process as with
;the Draw100 function but with 1 instead.
        Call    #DrawIntDirect
        Inc     R6
        Mov     #0,R8
        jmp     fin

isNegative:                             ;------Jomar Santos------
        cmp     #1,R9                       ;In subtraction, if
;the result is negative, then a 1 will be stored in R9.
        jeq    DrawNegative             ;If the condition is
;met, then the result is negative and we jmp to DrawNegative.
        jmp     fin

DrawNegative:   ;---Jomar Santos referencing Ignacio's code---
     BIS.B #0x04,&0xA2A                     ;Draw Negative
     jmp   fin

StartDisplay:   ;---Jomar Santos referencing Ignacio's code---
        Call #DrawDigit1                ;Calls function to
;write 0 on the first segment to initialize EnterInt process.
        jmp  fin

EnterInt:                               ;------Jomar Santos------
        mov.b   &P1IN,R13               ;In this whole
;function we loop and listen to when we have a left or right
;click. If the conditions are met, we jump to respective
;functions.
        and.b   #00000110B,R13
        cmp.b   #00000100B,R13
        jeq     nextDigit
        cmp.b   #00000010B,R13
        jeq     secondSetup
        jmp EnterInt

nextDigit:                              ;------Jomar Santos------
        INC     R5                          ;Increase R5 by one
;to display the next number option.

        cmp     #10,R5
        jeq     EqualTo10               ;If R5 is equal to 10
;we reset it to 0.
```

```
        Call    #SegDraw                ;We draw the number
;in R5, which is the next option.

        Call    #Delay                  ;We delay to
;accurately take in the data the user wants stored.
        Mov     #0,R13                  ;Reset R13 back to 0.
        jmp     EnterInt                ;jmp back to
;listening for left or right input.

EqualTo10:                          ;------Jomar Santos------
        Mov     #0,R5                   ;Resets R5 back to 0.
        Call    #SegDraw                ;Displays the 0 to
;the launchpad.
        Call    #Delay                  ;We delay to
;accurately take in the data the user wants to store
        jmp     EnterInt                ;jmp back to
;listening for left or right input.

secondSetup:                        ;------Jomar Santos------

        Call    #SaveInput              ;Stores the input the
;user provided.
        Call    #Delay
        inc     R15                     ;Increase R15 since
;this will keep track for the segments left to chose integers
;for to complete the integer.
        cmp     #3,R15
        jeq     termine                 ;If R15 is equal to 3
;then we finished picking the integer.
        Mov.w   #0,R13                  ;Resets R13, to
;listen for left and right input properly.
        Mov.w   #0,R5                   ;Resets R5 back to 0.
        Call    #SegDraw                ;Displays the 0 on
;the launchpad.
        jmp     EnterInt                ;jmp back to
;listening for left or right input.

termine:                            ;------Jomar Santos------
        Mov     R10,R9                  ;Move the stored
;integer value to R9.
        MOV.W   #0,R5                   ;Resets R5 to 0.
        CALL    #ClearLCD
        jmp     fin
```

```
SegDraw:                            ;------Jomar Santos------

        cmp     #0,R15                  ;Depending on the
;position stored on R15, we draw the integer on the
;corresponding segment from the 1st to the 3rd.
        jeq     DrawDigit1
        cmp     #1,R15
        jeq     DrawDigit2
        cmp     #2,R15
        jeq     DrawDigit3

SaveInput:                          ;------Jomar Santos------
        Mov     R5,R13                  ;This funtion saves
;the values inside R5 in R13, to properly save the wanted
;value of the user on the register.
        cmp     #0,R15
        jeq     Save1st
        cmp     #1,R15
        jeq     Save2nd
        cmp     #2,R15
        jeq     Save3rd

Save1st:                            ;------Jomar Santos------

        cmp     #0,R13
        jne     still1                  ;If we are on the 1st
;segment and R13 is not 0 we jmp to still1. R13 is our counter
;for the number.
        jmp     fin

still1:                             ;------Jomar Santos------
        add     #100,R10                ;Since we are on the
1st segment we add 100 to R10.
        dec     R13                     ;Here we decrease our
counter by 1.
        jmp     Save1st

Save2nd:                            ;------Jomar Santos------
        cmp     #0,R13                  ;Same as funtion
;Save1st but with the second segment
        jne     still2
        jmp     fin

still2:                             ;------Jomar Santos------
```

```
        add     #10,R10                 ;Same as funtion
;still1 but we add 10, since we are in the second segment
;which are multiples of 10.
        dec     R13
        jmp     Save2nd

Save3rd:                                ;------Jomar Santos------
        cmp     #0,R13                  ;Same as funtion
;Save1st but with the third segment
        jne     still3
        jmp     fin

still3:                                 ;------Jomar Santos------
        add     #1,R10                  ;Same as function
still1 but we add 1, since we are in the third segment which
are increments of one. Which I could have just used Inc
instead.
        dec     R13
        jmp     Save3rd

EnterSign:                              ;------Jomar Santos------
        mov.b   &P1IN,R13               ;Same as the EnterInt
;function but when left click cycles through the operations
;available (+,-,*,/) and if right clicks then it jumps to the
;next step.
        and.b   #00000110B,R13
        cmp.b   #00000100B,R13
        jeq     nextSign
        cmp.b   #00000010B,R13
        jeq     fin
        jmp     EnterSign

nextSign:                               ;------Jomar Santos------
        Call    #Delay
        cmp     #4,R7
        jge     strtAgain               ;If R7 is at 4 then
;we reset it back to 0 (+)
        call    #ClearLCD
        call    #DrawSign
        Mov     R7,R4
        inc.b   R7
        jmp     EnterSign

strtAgain:                              ;------Jomar Santos------
```

```
        mov.b   #0,R7                   ;Resets signs back to (+)
        jmp     nextSign

Operate:
        Mov     #0,R10
        Mov     #0,R12
        cmp     #0,R4
        jeq     sum
        cmp     #1,R4
        jeq     substraction
        cmp     #2,R4
        jeq     mult
        cmp     #3,R4
        jeq     divi

sum:
        add     R8,R9
        Mov     R9,R10
        jmp     fin

substraction:
        cmp     R8,R9
        jeq     equalToZero
        cmp     R8,R9
        jge     swap
        sub     R9,R8
        Mov     R8,R10
        Mov     #0,R8
        jmp     fin

equalToZero:
        Mov     #0,R10
        jmp     fin

swap:
        Mov     R9,R10
        Mov     R8,R9
        Mov     R10,R8
        sub     R9,R8
        Mov     R8,R10
        Mov     #1,R9
        jmp     fin

mult:
```

```
        cmp     #1000,R10
        jge     nextRegister
        cmp     #0,R9
        jeq     fin
        cmp     #0,R8
        jeq     fin
        dec     R9
        add     R8,R10
        jmp     mult

MoveValues:
        Inc     R12
        sub     #1000,R10
        jmp     mult

nextRegister:
        Inc     R12
        sub     #1000,R10
        jmp     mult

divi:
        cmp   #0,R9
        jeq   fin
        cmp   #0,R8
        jeq   fin
        sub   R9,R8
        jn    fin
        add   #1,R10
        jmp   divi

HoldResult:                             ;------Jomar Santos------
        mov.b   &P1IN,R13                   ;Displays result
until right click which starts the calculator process for next
operation if any.
        and.b   #00000110B,R13
        cmp.b   #00000010B,R13
        jeq     ClearReg                    ;When right clicked
;resets everything for next operation
     jmp     HoldResult

ClearReg:                               ;------Jomar Santos------
        Call    #Delay                      ;This function clears
;register for next run
        mov     #0,R4
```

```
        mov     #0,R5

        mov     #1,R6                   ;Tells us which
;segment we will use and we always at the very least use the
;1st. Because of this we pass a 1 to R6.

        mov     #0,R7
        mov     #0,R8
        mov     #0,R9
        mov     #0,R10
        mov     #0,R11
        mov     #0,R12
        mov     #0,R13
        mov     #0,R14
        mov     #0,R15
        Call    #ClearLCD

        jmp     fin

Delay:                                  ;------Jomar Santos------
        MOV     #50000,R13              ;This function passes
;a 50000 to R13 so that it functions as a delay to accurately
;carry out the functions of the calculator.

Next:
        dec     R13                     ;Here we loop through
;Next until R13 is 0.
        jnz     Next
        jmp     fin

;Objective: Draw an arithmetic sign (+, -, *, or /) determined
by R7 (0, 1, 2, or 3) at segment 1-6 on the LCD, determined by
R6
;Preconditions: Arithmetic sign must be on R7 (0-3), and
Segment must be on R6 (1-6)
;Postconditions: Arithmetic sign is drawn on the LCD. Nothing
has changed.
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawSign:
        push.w  R7 ;Save R7
        push.w  R5 ;Save R5
        add     #10,R7     ;Add 10 so 0->10, 1->11, 2->12 etc.
```

```
        mov.b   R7,R5    ;Move R7 to R5 so that we can draw R7
        call    #DrawDigit       ;Take us straight to the part
;of Drawint where we decide where to draw R5 which should be a
;digit but if it isn't it goes further in the array to find
;the signs.
        pop.w   R5 ;Return R5
        pop.w   R7 ;Return R7
        ret


;Objective: Draws a whole int (-10000 not included to 10000
not included) to the LCD starting at position 1-6 on the LCD
;Preconditions: Whole int must be on R5, and starting position
must be on R6
;Postconditions: Number is drawn on the LCD. Nothing has
changed.
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021
DrawIntSafe:

DrawInt:
       push.w   R6
       push.w   R11
       push.w   R5

    ;CHECK IF NEGATIVE AND IF NEGATIVE FLIP
      cmp.w     R5,0
      jlo       DrawIntFlipR5

DrawIntFlipR5Cont:
      call      #DrawIntDirect
      pop.w     R5
      pop.w     R11
      pop.w     R6

;CHECK IF NEGATIVE AGAIN AND IF NEGATIVE DRAW NEGATIVE SIGN.
      cmp.w     R5,0
      jlo       DrawIntDrawNegative    ;no need for a cont tag
;since we can use the return in DrawIntDrawNegative to return.
      ret

;Objective: Convert R5 from a negative value to a positive
value
;Preconditions: value to bit invert must be on R5
```

```
;Postconditions: R5's binary value is returned inverted, and
incremented by 1. (IE 0xFFFF --> 0x0001)
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntFlipR5:
      inv       R5
      inc.w     R5
      jmp       DrawIntFlipR5Cont

;Objective: Draws the negative sign on the first position
;Preconditions: None
;Postconditions: Negative sign is drawn.
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntDrawNegative:
      BIS.B     #0x04,&0xA2A ;Draw Negative
      ret ;return

;Objective: Draws an unsigned (assumed positive) integer at
position R6 WITHOUT ensuring that R5, R6, and R11 are
preserved. Should only be used internally by DrawInt
;Preconditions: Integerger value must be at R5 and must be
positive unsigned. Position value must be at R6 and must be
1-6
;Postconditions: int at R5 is drawn at position R6 on the LCD.
Registers may have been modified.
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntDirect:
      cmp.w     #7,R6
      jge       DrawIntOver10000 ;If R6 is over 6 (IE >=7)
;then oops that's beyond the screen RETURN

      cmp.w     #10000,R5
      jge       DrawIntOver10000 ;If it's over 10,000 return
;we aren't drawing that

      mov.b     #0,R11 ;reset R11 becuase we're going to use
;it to *count*

      cmp.w     #1000,R5
```

```
      jge       DrawIntOver1000 ;If it's over 1000

      cmp.w     #100,R5
      jge       DrawIntOver100 ;If it's over 100

      cmp.w     #10,R5
      jge       DrawIntOver10 ;If it's over 10
      ;Here we continue on to DrawDigit. This is intentional

;Objective: Draws digit or sign in R5 (0-13) at position R6
(1-6) on the LCD.
;Preconditions: Value to draw must be on R5, and position must
be on R6
;Postconditions: R5's value is drawn to R6 on the LCD. Nothing
is changed
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawDigit:
      ;Draw the specific digit
      cmp.b     #6,R6
      jeq       DrawDigit6

      cmp.b     #5,R6
      jeq       DrawDigit5

      cmp.b     #4,R6
      jeq       DrawDigit4

      cmp.b     #3,R6
      jeq       DrawDigit3

      cmp.b     #2,R6
      jeq       DrawDigit2

      cmp.b     #1,R6
      jeq       DrawDigit1

      ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(The following subroutines DrawDigit1 - DrawDigit6) can be
described using the following comment)
```

```
;Objectives: Draws digit or sign in R5 (0-13) at position N on
the LCD (Where N represents the number at the end of
DrawDigitN).
;Preconditions: Value to draw must be on R5
;Postconditions: R5's value is drawn to N on the LCD. Nothing
is changed
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawDigit1:
      MOV.B     #9,R11 ;Reuse R11 because why not
      MOV.B     DigitH(R5),0xA20(R11)
      MOV.B     DigitL(R5),0xA20+1(R11)
ret

DrawDigit2:
      MOV.B     #5,R11
      MOV.B     DigitH(R5),0xA20(R11)
      MOV.B     DigitL(R5),0xA20+1(R11)
ret

DrawDigit3:
      MOV.B     #3,R11
      MOV.B     DigitH(R5),0xA20(R11)
      MOV.B     DigitL(R5),0xA20+1(R11)
ret

DrawDigit4:
      MOV.B     #2,R11
      MOV.B     DigitH(R5),0xA30(R11)
      MOV.B     DigitL(R5),0xA30+1(R11)
ret

DrawDigit5:
      MOV.B     #14,R11
      MOV.B     DigitH(R5),0xA20(R11)
      MOV.B     DigitL(R5),0xA20+1(R11)
ret

DrawDigit6:
      MOV.B     #7,R11
      MOV.B     DigitH(R5),0xA20(R11)
      MOV.B     DigitL(R5),0xA20+1(R11)
ret
```

```
;;;;;;;;;;;;;;;; DRAWINT INTERNAL SUBROUTINES ;;;;;;;;;;;;;;;;

;The following subroutines are meant to be *internal* and
should not be used outside of DrawInt

;Objectives: Return call for values over 10000 or R6>7
;Preconditions: (none)
;Postconditions: Returns
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntOver10000:
      ret ;Return because we won't need to do that. ahahahaha

;Objectives: Handles the process of drawing an integer over
1000.
;Preconditions: Value to draw must be on R5 and must be at
least over 1000. R11 must already be 0
;Postconditions: R5's Thousands place is drawn to the LCD at
position R6. R5's thousands place is removed, and R6 is
incremented. R11's value is returned to 0 when done. Hands off
process to DrawIntOver100
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntOver1000:
      sub.w     #1000,R5 ;Subtract and
      inc.b     R11    ;keep track of how many times
;subtracted 1000
      cmp       #1000,R5
      jge       DrawIntOver1000  ;until it is no longer >=1000

     ;Draw and reset R11
      call      #DrawIntDrawR11

      inc.b     R6 ;Increment R6

      jmp       DrawIntOver100;Continue on to DrawIntOver100

;Objectives: Handles the process of drawing an integer over
100.
;Preconditions: Value to draw must be on R5. R11 must already
be 0
```

```
;Postconditions: R5's Hundreds place is drawn to the LCD at
position R6. R5's Hundreds place is removed, and R6 is
incremented. R11's value is returned to 0 when done. Hands off
process to DrawIntOver10
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021
DrawIntOver100:

      cmp       #100,R5
      jlo       DrawIntUnder100Already  ;Check if it's bellow
100. Otherwise,

      sub.w     #100,R5 ;Subtract and
      inc.b     R11    ;keep track of how many times
subtracted 100
      jmp       DrawIntOver100  ;until its no longer >=100

      ;Internal jump point to indicate we're already under
100.
      DrawIntUnder100Already:

      ;Draw and reset R11
      call      #DrawIntDrawR11
      inc.b     R6 ;Increment R6

      jmp       DrawIntOver10 ;Continue on to DrawIntOver10

;Objectives: Handles the process of drawing an integer over
10.
;Preconditions: Value to draw must be on R5. R11 must already
be 0
;Postconditions: R5's tens place is drawn to the LCD at
position R6. R5's tens place is removed, and R6 is
incremented. R11's value is returned to 0 when done. Hands off
process to DrawDigit
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntOver10:
      cmp       #10,R5
      jlo       DrawIntUnder10Already  ;Check if we're already
;under 10. Otherwise...

      sub.w     #10,R5 ;Subtract and
```

```
      inc.b     R11    ;keep track of how many times
;subtracted 1000
      jmp       DrawIntOver10  ;until its no longer >=10

;Internal jump point to indicate we're already under 10

DrawIntUnder10Already:
      ;Draw and reset
      call      #DrawIntDrawR11
      inc.b     R6 ;Increment R6

      ;Jump to DrawDigit. It has a return so we'll return once
;we draw the last digit
      jmp       DrawDigit


;Objectives: Handles the process of drawing a digit that's in
R11 and clearing R11. INTERNAL USE ONLY FOR DrawIntOver(N)
SUBROUTINES.
;Preconditions: Value to draw must be in R11 and must be
between 0-9. Position to draw R11 must be on R6
;Postconditions: R11 is drawn to position R6, and is then set
to 0. R5 is preserved.
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

DrawIntDrawR11:
      push.w    R5 ;Save R5
      mov.w     R11,R5 ;Move R11 to R5
      call      #DrawDigit ;Draw R11 (now in R5)
      pop.w     R5 ;Return R5
      mov.b     #0,R11 ;Reset R11 for the next subroutine.
      ret

;Objectives: Clears the LCD
;Preconditions: LCD is already initialized
;Postconditions: LCD is cleared
;Author: Ignacio Tampe (igtampe)
;Date: 3/7/2021

ClearLCD:
      MOV.W   #2,&LCDCMEMCTL
      BIS.W   #1,&LCDCCTL0
      ret
```

```
fin:    RET

TheEnd: JMP $                           ; jump to current
location '$'
                                        ; (endless loop)
        END
```

Documentación de las subrutinas

DrawInt

- Objetivo: Dibujar en la pantalla del launchpad el número deseado.
- Precondiciones: R5 debe contener en número el cual desea ser imprimido.
- Postcondiciones: El launchpad, mostrará en su pantalla el número indicado por R5.
- Autor: Ignacio Tampe
- Fecha: 6/Marzo/2021

DrawSign

- Objetivo: Dibujar en la pantalla del launchpad la operación indicada por R7 (+,-,*,/)
- Precondiciones: R7 debe se mayor a 0 y menor a 4.
- Postcondiciones: Al incrementar R7 a un valor mayor a 3, al mismo será pasado un valor de 0.
- Autor: Ignacio Tampe
- Fecha: 6/Marzo/2021

Add

- Objetivo: Sumar dos números, ubicados en los registros R8 y R9 y ubicar su resultado en R10.
- Precondiciones: R8 y R9 ambos contienen dígitos para realizar la suma.
- Postcondiciones: R10 Contiene el resultado de la suma.
- Autor: Peter Santana
- Fecha: 5/Marzo/2021

Subtract

- Objetivo: Restar dos números, ubicados en los registros R8 y R9 y ubicar su resultado en R10.
- Precondiciones: R8 y R9 ambos contienen dígitos para realizar la resta. R8 es mayor que R9, si no, se utiliza la subrutina SWAP.
- Postcondiciones: R10 Contiene el resultado de la resta.
- Autor: Peter Santana
- Fecha: 5/Marzo/2021

Multiply

- Objetivo: Realizar la operación de multiplicación utilizando los valores ubicados en los registros R8 y R9.
- Precondiciones: R8 y R9 deben contener los dígitos, los cuales serán utilizados para realizar la operación. De alguno de los dos registros contener 0, la operación culminará con un resultado de 0 en R10.
- Postcondiciones: De ser el resultado mayor a 1000, el valor sera guardado en dos registros. R12 contendrá los números que indican los miles, y R10 las centenas. Por ejemplo, de ser el resultado 998001, R12 contendra 998 y R10 contendra 001.
- De el resultado obtenido ser menor a 1000, el mismo se guardará dentro de R10.
- Autor: Jomar Santos
- Fecha: 10/Marzo/2021

Divide

- Objetivo: Dividir dos números, ubicados en los registros R8 y R9 y ubicar su resultado en R10.
- Precondiciones: R8 y R9 ambos contienen dígitos para realizar la división. R8 es mayor que R9, si no, se utiliza la subrutina SWAP.
- Postcondiciones: R10 Contiene el resultado de la división.
- Autor: Peter Santana
- Fecha: 5/Marzo/2021

Operate

- Objetivo: Dirigir a la operación deseada utilizando los valores ubicados en los registros R8 y R9. Una vez la misma sea resuelta, devolver los resultados en el R10, y de ser multiplicación con resultado mayor a 1000, en los registros R12 y R10.
- Precondiciones: R8 y R9 deben contener los dígitos, los cuales serán utilizados para realizar la operación. Además, se debe haber escogido una operación a realizar. Una vez estos sean proveídos, se hace un reset a R10 y R12 como edge case, para qué hace guarden los resultados de las operaciones.
- Postcondiciones: El resultado de la operación será guardado en R10 de ser cualquier operación y multiplicación menor a 1000. De ser una multiplicación mayor a 1000, parte del resultado se guardará en dos registros, R12 y R10.
- De el resultado obtenido ser menor a 1000, el mismo se guardará dentro de R10.
- Autor: Jomar Santos
- Fecha: 6/Marzo/2021

EnterInt

- Objetivo: “Listen” por un input de left o right click, del usuario de la calculadora. De hacer un left click, imprime en la pantalla el próximo dígito, hasta llegar a 9. De hacer

un left click y estar en 9, el dígito que se imprimirá será 0. De el usuario hacer un right click, pues se procederá a imprimir al próximo segmento, y el dígito escogido será guardado en el registro correspondiente. Esto sucederá hasta haber elegido 3 dígitos.

- Precondiciones: Registro R9 debe estar vacío para así poder guardar el número elegido durante el proceso.
- Postcondiciones: Se guardará el dígito elegido por el usuario en el registro R9.
- Autor: Jomar Santos
- Fecha: 5/Marzo/2021

EnterSign

- Objetivo: "Listen" por un input de left o right click, del usuario de la calculadora. De hacer un left click, imprime en la pantalla el siguiente signo de operación, hasta llegar a "/". El mismo se lleva a cabo en el siguiente order: +, -, *, /. Una vez se presione el left click y el signo este en "/", el ciclo comenzará nuevamente.
  De el usuario hacer un right click, el signo elegido será guardado para uso como identificador de operación cuando el problema matemático se esté resolviendo.

- Precondiciones: Al llamar la función, R7 debe comenzar en 0. Así siempre empezamos imprimiendo el signo de "+".
- Postcondiciones: Se guardará el signo elegido por el usuario en el registro R4.
- Autor: Jomar Santos
- Fecha: 5/Marzo/2021

Main Routine

- Objetivo: Comenzar el programa de la calculadora, llamando así las funciones apropiadas. Una vez un proceso de calculadora sea realizado, el mismo comenzará nuevamente.
- Precondiciones: Todos los registros deberán contener 0 con la excepción de R6 que debe tener un 1, para comenzar debidamente el proceso de la calculadora.
- Postcondiciones: Se ejecutará satisfactoriamente el proceso de la calculadora, permitiendo así, elegir dos números y la operación a realizar, imprimiendo en la pantalla el resultado de la misma al culminar el proceso.
- Autor: Jomar Santos
- Fecha: 11/Marzo/2021

[1] Referencias

https://www.youtube.com/watch?v=2_ClFb73M5Y

https://www.youtube.com/watch?v=VeqNpyb8ftA

https://www.youtube.com/watch?v=PH2wOAzNQX4

https://www.youtube.com/watch?v=TcrGyMA3aT8