INFORME LABORATORIO 5

PRESENTADO POR:

JOSE DAVID GONZALEZ HENAO

LUIS EDUARDO RODRIGUEZ A

PRESENTADO A:

PROF: LUIS GERMAN GARCIA MORALES

LABORATORIO DE ELECTRONICA DIGITAL II

UNIVERSIDAD DE ANTIOQUIA

INGENIERIA ELECTRONICA

MEDELLIN-ANTIOQUIA

2019

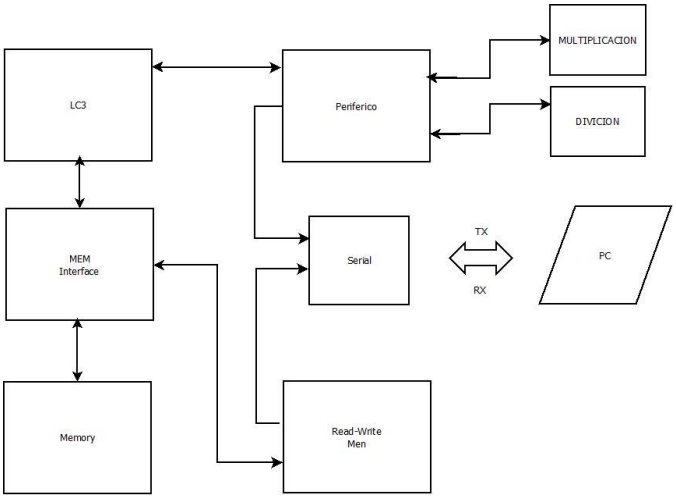
Resumen:

Para esta práctica se realizó la implementación de una calculadora con notación polaca inversa utilizando lenguaje ensamblador para el procesador LC3 implementado en la FPGA. Para llevar a cabo esto, se utilizó códigos anteriores y posteriormente algunos modificarlos para una correcta sintonía, (practica 4 y 2), tales como el módulo de multiplicación en vhdl, el código suministrado por el profesor del procesador lc3 y la comunicación serial y el código para la captura de datos en asembler. El código en asembleer se modificó de tal manera que despliegue un nuevo menú para el usuario, para el cual escoja que tipo de operación matemática quiera hacer, al igual que los números para tales operaciones; a diferencia del código anterior estos datos se almacenan en una pila de posiciones de memoria llamada stack donde pueden ser extraídos o proporcionados por las subrutinas push o pop. Ya con los operadores y operandos almacenados en el stack se procede a llamar las subrutinas para multiplicación, división, resta, suma y modulo y mostrar el resultado en pantalla.

Por otro lado el código de la multiplicación se modificó de tal manera que solo recibe números enteros positivos y negativos y retorne un resultado, y se acoplo con el modulo del lc3 proporcionado por el profesor, directamente en el submodulo peripheral, además para la operación de división y modulo se creó un nuevo código que igualmente al de multiplicacion se acoplo al código del LC3.

Finalmente se procede a conectar los módulos de multiplicación y división anteriormente mencionados de tal manera que desde el código en asembler se pueda enviar los operandos y estos módulos retornen una respuesta.

Implementación:



Para acceder a los módulos de multiplicación y división para posteriormente pasarle los datos u operandos se utilizó las siguientes subrutinas :

```
MULTIPLICAR     LD R2,DIR_MULT_A
                LD R3,DIR_MULT_B
                STR R7,R2,#0
                STR R6,R3,#0
WAIT_MULT       LDI R2,ESTADO_MULT
                BRZP WAIT_MULT
                LDI R2,RESULT1_MULT
                LDI R3,RESULT2_MULT
                STR R2,R4,#0
                BR NEXT_OPERACION

DIVIDIR         LD R2,DIR_MULT_A
                LD R3,DIR_DIV_B
                STR R7,R2,#0
                STR R6,R3,#0
WAIT_DIV        LDI R2,ESTADO_DIV
                BRZP WAIT_DIV
                LDI R3,RESULT_MAYOR
                LDI R2,RESULT_MENOR
                STR R2,R4,#0
                BR NEXT_OPERACION
```

Para almacenar los datos se creó dos stack, uno para los operadores y otro para operandos por lo cual en el codigo se muestra unas líneas donde se elige en cuál de los almacenarlos.

```
PUSH            LD R0,SELEC_STACK
                BRZ STACK_NUM
                LD R0,STACK_OPER
                LD R3,N2
                BRNZP GUARDAR2
STACK_NUM       LD R0,NUMVECT;----------
                LD R3,N1
GUARDAR2        ADD R5,R0,R3
                STR R4,R5,#0; Almacenamos el datos en la direccion (Numvect o stack_oper)
                ADD R3,R3,#1
                LD R0,SELEC_STACK
                BRP GUARDAR_OPER
                ST R3,N1
                BRNZP NEX_DATA
GUARDAR_OPER    ST R3,N2
                BRNZP NEX_DATA
```

Para realizar las operaciones primero se carga las direcciones de las pilas y posteriormente los datos contenidos en éstas. Con los datos de la pila de operadores se verifica que operación resulta y se procede a ejecutar la subrutina correspondiente.

```
OPERACIONES     ST R7,RETORNO_OPER
                LD R0,STACK_OPER        ;R0=Dir del stack de los operadaores
                LD R1,NUMVECT                   ;R1=Dir del stack de los numeros
                ADD R1,R1,#-1
NEXT_OPERACION          LD R3,N1
                ADD R5,R3,R1            ;apuntador al ultimo numero ;R5=N1+(NUMVECT)
```

```
ADD R4,R5,#-1          ;apuntador
ADD R3,R3,#-1
BRNZ VERI_OPER
LD R2,N2
BRNZ RESULTADO
LDR R6,R4,#0           ;Cargamos los datos
LDR R7,R5,#0
ADD R5,R5,#-1
LD R2,N1
ADD R2,R2,#-1
ST R2,N1
LDR R2,R0,#0           ;R2=dato(Operador)
ADD R0,R0,#1
LD R3,N2
ADD R3,R3,#-1
ST R3,N2
LD R3,MAS
ADD R3,R3,R2
BRZ SUMAR
LD R3,MENOS
ADD R3,R3,R2
BRZ RESTAR
LD R3,MULT
ADD R3,R3,R2
BRZ MULTIPLICAR
LD R3,DIV
ADD R3,R3,R2
BRZ DIVIDIR
LD R3,MOD
ADD R3,R3,R2
BRZ MODULO
BR RESULTADO
```

Conclusiones:

Se nota que hay mayor libertad al trabajar en la LC3 de la FPGA en lugar del simulador, ya que se tienen disponibles todas las posiciones de memoria menos las 4 de comunicación con los periféricos lo que permite mayor capacidad de almacenamiento, claro está que se deben implementar las funciones del LC3 que necesitemos, ya que en la FPGA sólo están las instrucciones básicas (sin los TRAP), esto requiere un pequeño esfuerzo mental extra pero es un costo que se debe pagar por la libertad de uso.
Según el análisis anterior para nuestro programa hay disponibles FFFF-(Número de líneas del programa + 4) posiciones de memoria para los datos numéricos ingresados por el usuario. Esto se puede implementar poniendo el BLKW en el final del código y se accede a él cargando en un registro el valor de la posición de memoria donde empieza y se hacen las demás funciones de manera normal.