UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS, QUÍMICAS Y NATURALES

Arquitectura de Computadoras

TRABAJO PRÁCTICO N° 3 :

BIP (Basic Instruction-set Processor)

ALUMNOS:

- BOSACK,Federico
- OLIVA, Nahuel

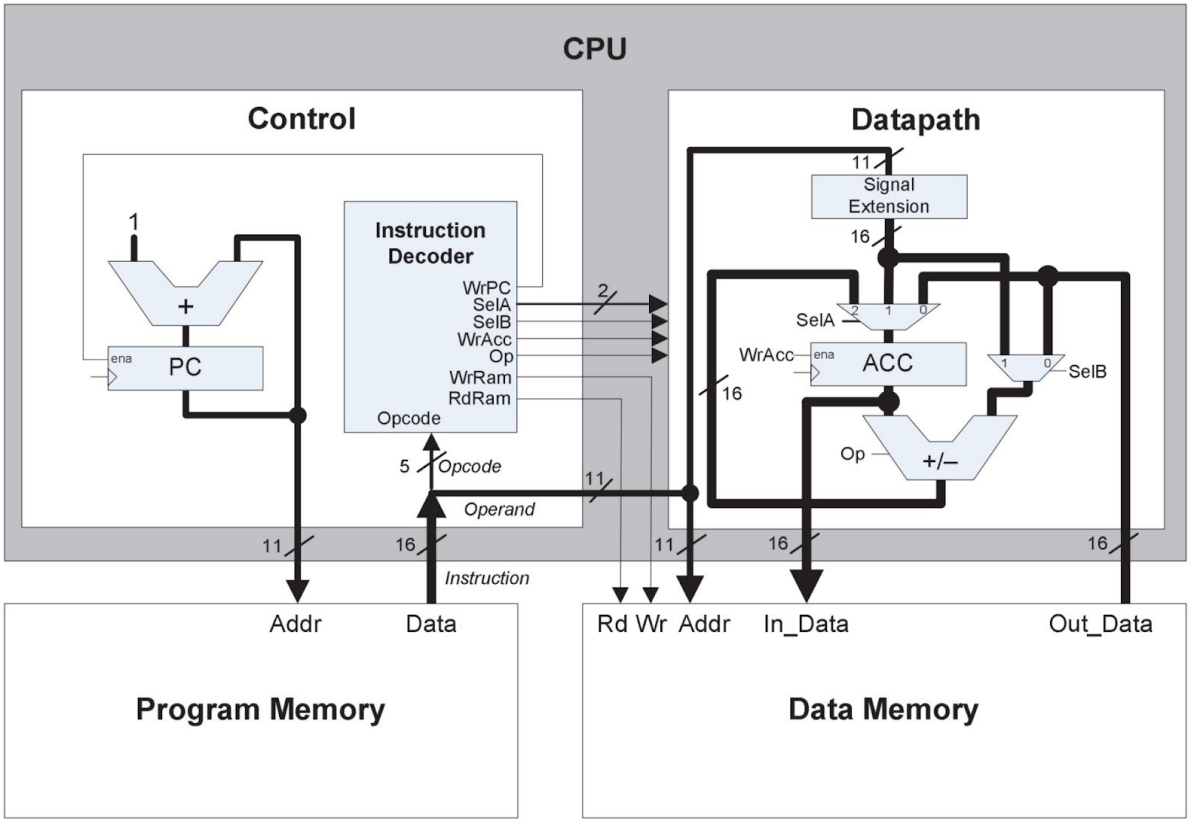
Fecha: 27/11/2019

INTRODUCCIÓN

El presente informe se realiza con el objetivo de describir en detalle la implementación de **procesador simplificado** (BIP : Basic Instruction-set Processor), con una arquitectura que, a primera vista, no tiene una gran diferencia con otros procesadores ,sin embargo es distinguido de los otros procesadores porque es diseñado utilizando un enfoque más interdisciplinario,el mismo se realizará utilizando el lenguaje de descripción de hardware, Verilog, a través de la IDE VIvado, junto con la placa FPGA, Basys 3.

Este procesador luego es interconectado con el módulo de Uart ya descrito en trabajo anterior ,el cual facilita una **interfaz** de control para la **inicialización** del mismo, permitiendo así la comunicación hacia un ordenador u otro dispositivo que implemente Uart.

A continuación se muestra el **diagrama de bloques** del procesador a implementar ,junto con la memoria de programa y de datos utilizada para la obtención de instrucciones y almacenamiento y lectura de datos.



El procesador deberá ser **monociclo**,es decir se tomará exactamente un ciclo de reloj para la ejecución de cualquier instrucción.

A continuación se muestran las **instrucciones** y códigos respectivos **soportados** por el mismo.

TABLE I. INSTRUCTION SET

Operation	Opcode	Instruction	Data Memory (DM) and Accumulator (ACC) Updating	Program Counter (PC) updating
Halt	00000	**HLT**		PC ← PC
Store Variable	00001	**STO** operand	DM[operand] ← ACC	PC ← PC + 1
Load Variable	00010	**LD** operand	ACC ← DM[operand]	PC ← PC + 1
Load Immediate	00011	**LDI** operand	ACC ← operand	PC ← PC + 1
Add Variable	00100	**ADD** operand	ACC ← ACC + DM[operand]	PC ← PC + 1
Add Immediate	00101	**ADDI** operand	ACC ← ACC + DM	PC ← PC + 1
Subtract Variable	00110	**SUB** operand	ACC ← ACC - DM[operand]	PC ← PC + 1
Subtract Immediate	00111	**SUBI** operand	ACC ← ACC - operand	PC ← PC + 1

Formato de la instrucción del BIP I:

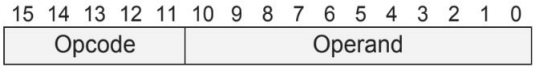


Figure 1. Instruction format

● **Opcode:** campo de 5 bits de longitud que identifica la operación a ser realizada por la instrucción.

● **Operando**: campo de 11 bits de longitud para identificar el operando de la instrucción. Éste puede representar una constante (dato inmediato) o una dirección de la memoria de datos(variable).

Registros de la CPU del BIP I:

● PC: el contador de programa. Almacena la dirección de la instrucción actual.

● ACC: el acumulador. Trabaja como un operando implícito en las instrucciones.

Organización de la memoria:

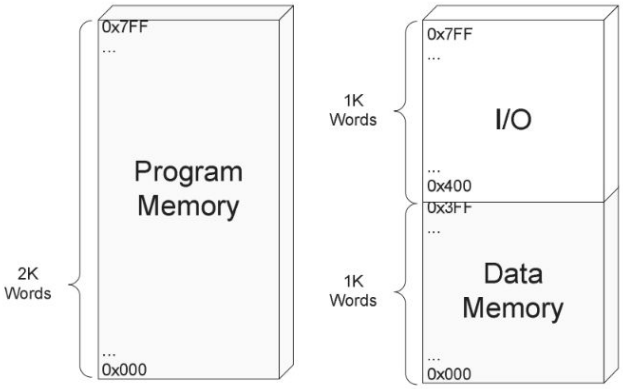
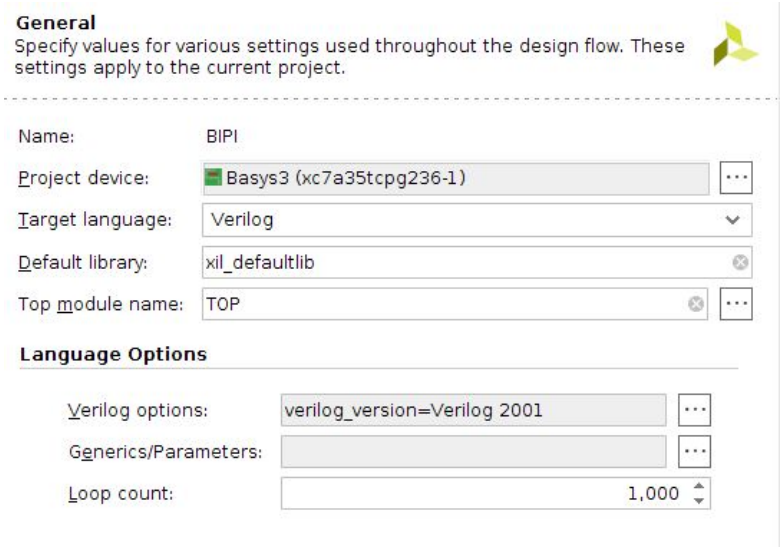


Figure 2. Addressing spaces

- Memoria de programa: 2 kb.
- Memoria de datos: 1 kb.
- Memoria I/O: No utilizada.

DESARROLLO

Para la implementación del sistema se utilizó el entorno de desarrollo IDE **Vivado** con las siguientes configuraciones.



Módulo principal TOP.

Se definió un módulo Top, el cual encapsula el sistema entero ofreciendo un nivel de abstracción del funcionamiento interno.

El mismo consta de tan solo tres entradas ,el clock,el hardware reset , el receptor uart rx(el cual se optó por usar el uart integrado dentro del conector microUsb de la Fpga) y una salida para la muestra de los valores de la parte baja de los registros (CC y ACC) al finalizar la ejecución de las instrucciones cargadas previamente en la memoria de programa a los leds internos de la placa.

Este módulo principal integra los siguientes subsistemas :

- receptor uart.
- baud Rate Generator.
- Interface.
- control.
- datapath.
- Memoria de programa.
- Memoria de datos.
- CC (contador de ciclos).

Generador de Ticks para el BaudRate

El generador de velocidad en baudios genera una señal de muestreo cuya frecuencia es exactamente 16 veces la velocidad de transmisión designada del UART. Para evitar crear un nuevo dominio de reloj y violar el Principio de diseño síncrono, la señal de muestreo se realiza mediante la generación de ticks en lugar de la señal de reloj.

Para la velocidad de 19.200 baudios, la velocidad de muestreo debe ser de 307.200 (es decir, 19.200 * 16) ticks por segundo. Como la velocidad del reloj del sistema es de 50 MHz, el generador de velocidad en baudios necesita generar un tick cada 163 ciclos de reloj.

$$\frac{Clock}{BaudRate * 16} \cong 163$$

Recepción Uart

El siguiente módulo se utilizó para la recepción y sincronización para la señal de start (ff) este incluye cuatro estados principales,idle, start, data y stop, que representan el procesamiento del bit de inicio, bits de datos y de parada. La señal s_tick es la señal de activación del generador de velocidad de transmisión y hay 16 ciclos en un intervalo de bits, hay que tener en cuenta que el FSMD permanece en el mismo estado a menos que la señal s_tick sea positiva. Hay dos contadores, representados por los registros contador_ticks y contador_bits. El registro contador_ticks mantiene un seguimiento del número de ticks de muestreo y cuenta hasta 7 en el estado start, hasta 15 en el estado data y CANT_BIT_STOP en el estado stop. El registro contador_bits realiza un seguimiento de la cantidad de datos bits recibidos en el estado data. Los bits recuperados se desplazan y se vuelven a montar en el registro de buffer. Se incluye una señal de estado, rx-done. La cual se activa luego de un ciclo de reloj después de que se completa el proceso de recepción.

Interface

Este módulo se utilizó para la detección de la señal de start (0xff) para proceder a dar inicio a la ejecución mediante la señal de reset compartida por todos los módulos secuenciales y dando inicio al program counter del procesador.

Contador de Ciclos (CC)

El módulo contador de ciclos consiste en un contador de pulsos de clock para la validación de la propiedad monociclo del procesador,la cual se logró combinando el uso tanto de los flancos descendentes (para la actualización del program counter y lectura/escritura en la memoria de datos) como los flancos ascendentes (para la obtención de la instrucción y actualización del registro Acc y CC), desde que llega la señal de inicio (ff)al receptor Uart hasta que el procesador ejecute la instrucción HALT.

Módulo de Control

A fin de seguir el diagrama de bloques presentado en la introducción, se implementa el módulo de control ,el cual integra los siguientes módulos :

- Contador de programa.
- decodificador de instrucciones.

Contador de programa (Pc)

El siguiente módulo se utilizó con el propósito de generar un sumador (11 bits) para la obtención de la dirección de la instrucción siguiente ya almacenada en la memoria de programa.

Decodificador de instrucciones

El siguiente módulo se utiliza para recibir el opcode de la instrucción leída de la memoria de programa y mediante un circuito combinacional se configuran los valores de las señales de control, para el direccionamiento al módulo Datapath ,para lectura/escritura en el caso de las memorias y para la desactivación del program counter (wrPc para el caso de la instrucción Halt).

Datapath

El módulo Datapath posee el registro ACC y toda la lógica para poder realizar la operación especificada por la instrucción. Dicha lógica consiste en:

● Un sumador/restador de 16 bits que permite realizar las operaciones de adición y resta solicitadas por las instrucciones ADD, ADDI, SUB y SUBI.

● Un módulo de extensión de signo de 11 bits a 16 bits. Esto es porque existe una diferencia de bits entre un operando obtenido de la memoria, el cual es de 16 bits, y uno presente en la instrucción, el cual posee 11 bits.

● 2 multiplexores, los cuales son controlados por señales que provienen del módulo decodificador de instrucciones(selA y selB). Estos multiplexores controlan cuáles serán los operandos aritméticos o cuál será el valor del ACC en función de la operación a realizar.

Memorias

Para el módulo de memoria se utilizó un registro de datos de 1024/2048 (datos/programa) entradas de 16 bits (ancho de las instrucciones y datos) ,además se implementó un parámetro de inicialización en el cual se especifica la dirección del archivo de inicialización (txt) el cual permite cargar en el caso de la memoria de programa directamente el archivo binario de las instrucciones a ejecutar mediante el uso de la funcion $readmemh,en el caso de la memoria de datos este parámetro se lo deja en blanco,lo cual producirá una inicialización de los datos en 0 .

Cabe destacar que para el caso de la memoria de programa la lectura está sincronizada mediante los flancos descendentes del clock a diferencia de la memoria de datos la cual se

lee/escribe en flancos ascendentes,ya que es necesario para que las instrucciones puedan ejecutarse en sólo un ciclo,esta característica se la especifica a la hora de instanciar el módulo mediante otro parámetro.

TESTBENCHES

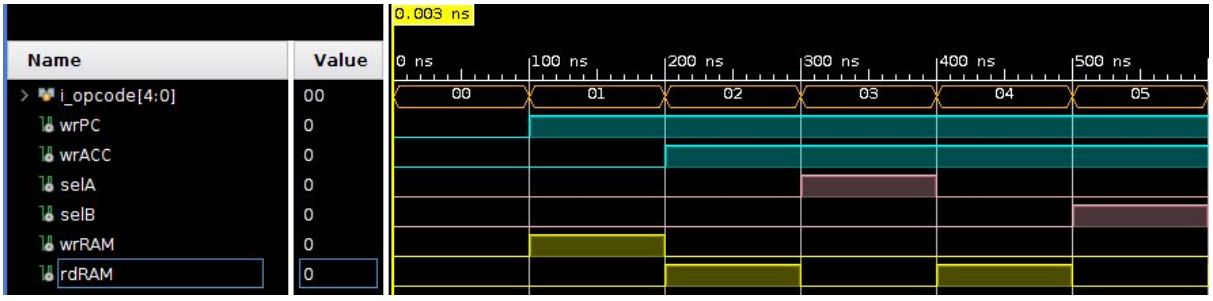
Para la validación del funcionamiento del sistema se realizaron cinco testbenches de manera progresiva de los siguientes subsistemas :

1. TestBench del decodificador de instrucciones
2. Read/Write Memory TestBench
3. Datapath Testbench
4. Interface TestBench
5. Top TestBench

TestBench del decodificador de instrucciones

En la siguiente gráfica se muestra el análisis del módulo decodificador de instrucciones ,validando cada una de ellas.

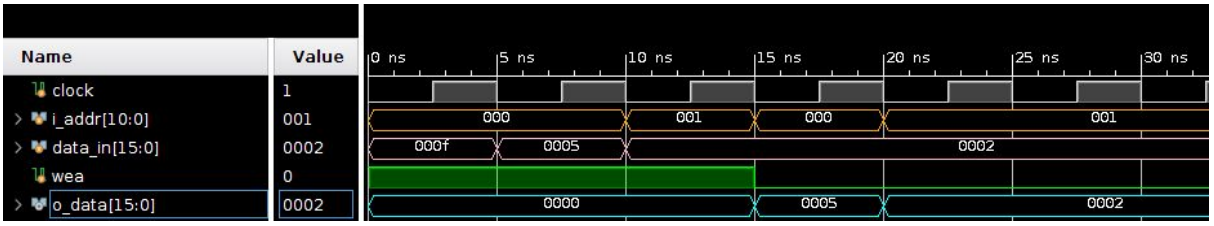
Se puede observar el correcto funcionamiento de la lógica combinacional del mismo,verificado por las seis señales de control,wrPc,wrAcc,selA,selB y wr/rdRam.



Read/Write Memory TestBench

A continuación se observa el testbench para validar los módulos de memoria,realizando una escritura,sobre escritura y lectura de datos.

```
initial    begin

    clock = 1'b0;
    reg_i_addr = 11'b0000000;  //escribo 16 en dir0
     data_in = 8'b00001111;
    reg_wea = 1'b1;
    #5

    data_in = 8'b00000101; //escribo 5 en dir0
    reg_wea = 1'b1;
    #5

    reg_i_addr = 11'b0000001; //escribo 2 en dir1
    data_in = 8'b00000010;
    reg_wea = 1'b1;
    #5

    reg_wea = 0'b0;
    reg_i_addr = 11'b0000000; // Lectura de la dir0
    #5
    reg_i_addr = 11'b0000001; // Lectura de la dir1
    #5

    #500000 $finish;
end
```



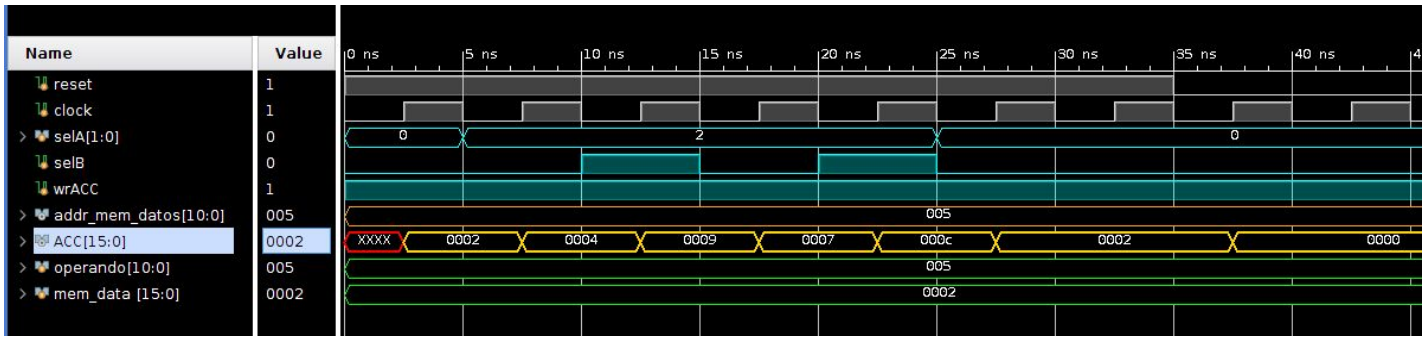
En el gráfico de arriba se pueden distinguir las distintas acciones en memoria.

a) Escritura 000f en dirección de memoria 0 . (Intervalo 0-5ns)
b) Sobre escritura 0005 en dirección de memoria 0 . (Intervalo 5-10ns)
c) Escritura 0002 en dirección de memoria 1 . (Intervalo 10-15ns)
d) Lectura 0005 en dirección de memoria 0. (Intervalo 15-20ns)
e) Lectura 0002 en dirección de memoria 1. (Intervalo 20-35ns)

Datapath Testbench

A continuación se muestra el código implementado para la validación del datapath,el mismo consta de un valor de operando y dato de lectura en memoria fijo, para verificar la evolución del registro acumulador a la hora de ejecutar distintas instrucciones.

```
initial begin
        reg_clock = 1'b0;
        reg_reset = 1'b0;

        reg_dato_mem = 2;
        reg_operando = 5;
        reg_reset = 1'b1; // comienzo el PC.

reg_wrACC = 1;//escribo al registro acc
reg_selA = 0;//entrada al ACC <- dato en mem
reg_selB = 0;//segundo operando alu <- dato en mem
 reg_i_opcode = 2; //Load variable. (2)
#5

reg_wrACC = 1;//escribo al registro acc
reg_selA = 2;//escribo al ACC la salida de la alu
reg_selB = 0;//2do operando de la alu > salida mem de datos
reg_i_opcode = 4; //Add variable. (2)
#5

reg_wrACC = 1;//escribo al registro acc
reg_selA = 2;//escribo al ACC la salida de la alu
reg_selB = 1;//2do operando de la alu > salida mem de datos
reg_i_opcode = 5; //Add inmidiate
#5

reg_wrACC = 1;//escribo al registro acc
reg_selA =2;//escribo al ACC la salida de la alu
reg_selB = 0;//2do operando de la alu > salida mem de datos
reg_i_opcode = 6; //Sub variable
#5

reg_wrACC = 1;//escribo al registro acc
reg_selA = 2;//escribo al ACC la salida de la alu
reg_selB = 1;//2do operando de la alu > salida mem de datos
reg_i_opcode = 5; //Add inmidiate
#5

reg_wrACC = 1;//escribo al registro acc
reg_selA = 0;//entrada al ACC <- dato en mem
reg_selB = 0;//segundo operando alu <- dato en mem
 reg_i_opcode = 2; //Load variable. (2)
#10

       reg_reset = 1'b0; // Reset.
        #50 reg_reset = 1'b1; // Desactivo el reset.

        #1000000 $finish;
end
```



En el gráfico de arriba se pueden verificar las siguientes operaciones

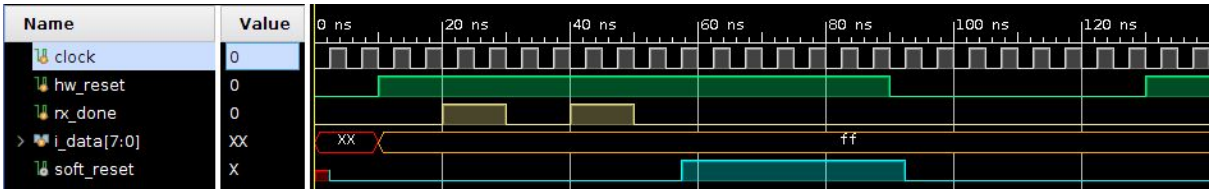
a)Load-Var (Acc=2) (2,5 ns) **b)**Add-Var(Acc+2=4) (7,5 ns) **c)** Add-I (Acc+5=9) (12,5 ns)

d) Sub-Var (Acc-2 = 7) (17,5 ns) **e)** Add-I (Acc+5 = C) (22,5 ns) **f)** Load-Var (Acc=2) (27,5 ns)

Y luego finalmente a los 35ns se realiza un hardware reset del sistema,teniendo efecto a los 37,5ns al salto positivo del clock,dejando en 0 al registro acumulador.

Interface TestBench

A continuación se observa el accionamiento del soft reset al detectar las señales de start, el cual dará lugar al comienzo del procesador mediante la activación del contador de programa y funcionamiento de los registros del sistema.



En el gráfico se pueden distinguir los siguientes intervalos :

a)Recepción del 1er comando de start (~25ns),a través de la señal de fin de recepción del módulo Rx.
b)Recepción del 2do comando de start(~45ns)

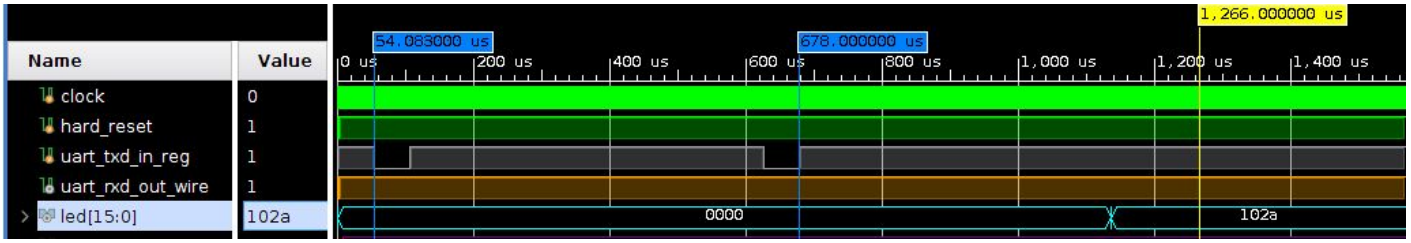
c)Accionamiento del sw reset (~55ns) **d)**hw reset(~90ns)

Top Testbench

Finalmente se realiza la prueba para el módulo principal del sistema,a continuación se muestran las instrucciones cargadas en la memoria de programa del procesador,junto con el cálculo de los registros para la verificación del testbench.

```
LDI 15     //ACC = 15 ; CC = 1;
STO 0      //ACC = 15 ; DM[0] = 0; CC = 2;
LD 0       //ACC = 15 ; DM[0] = 15; CC = 3;
ADDI 1     //ACC = 15 ; DM[0] = 15; CC = 4;
STO 1      //ACC = 16 ; DM[0] = 15;DM[1] = 0; CC = 5;
LD 1       //ACC = 16 ; DM[0] = 15;DM[1] = 16; CC = 6;
SUBI 1     //ACC = 16 ; DM[0] = 15;DM[1] = 16; CC = 7;
SUB 0      //ACC = 15 ; DM[0] = 15;DM[1] = 16; CC = 8;
ADD 1      //ACC = 0 ; DM[0] = 15;DM[1] = 16; CC = 9;
ADDI 10    //ACC = 16 ; DM[0] = 15;DM[1] = 16; CC = 10;
STO 2      //ACC = 26 ; DM[0] = 15;DM[1] = 16;DM[2] = 0; CC = 11;
LDI 0      //ACC = 26  ; DM[0] = 15;DM[1] = 16;DM[2] = 26; CC = 12;
LD 2       //ACC = 0 ; DM[0] = 15;DM[1] = 16;DM[2] = 26; CC = 13;
SUB 1      //ACC = 26 ; DM[0] = 15;DM[1] = 16;DM[2] = 26; CC = 14;
ADDI 32    //ACC = 10 ; DM[0] = 15;DM[1] = 16;DM[2] = 26; CC = 15;
HALT       //ACC = 42 (2a) CC = 16 (10)
```

Como se puede observar en la siguiente imagen ,al terminar de detectar el código del start (0xff) el procesador comienza la ejecución del programa y finalmente se distingue que la parte baja del registro acumulador (salida a parte baja de Leds) y contador de programa (salida a parte alta de Leds) ,coinciden con los valores calculados anteriormente,por lo que la ejecución de las 16 instrucciones fue exitosa.



Para más detalle,en la siguiente figura se realiza una acercamiento para observar cómo los valores de los registros van cambiando a medida que el procesador ejecuta las instrucciones individualmente.Los cuales coinciden nuevamente con los mencionados anteriormente.