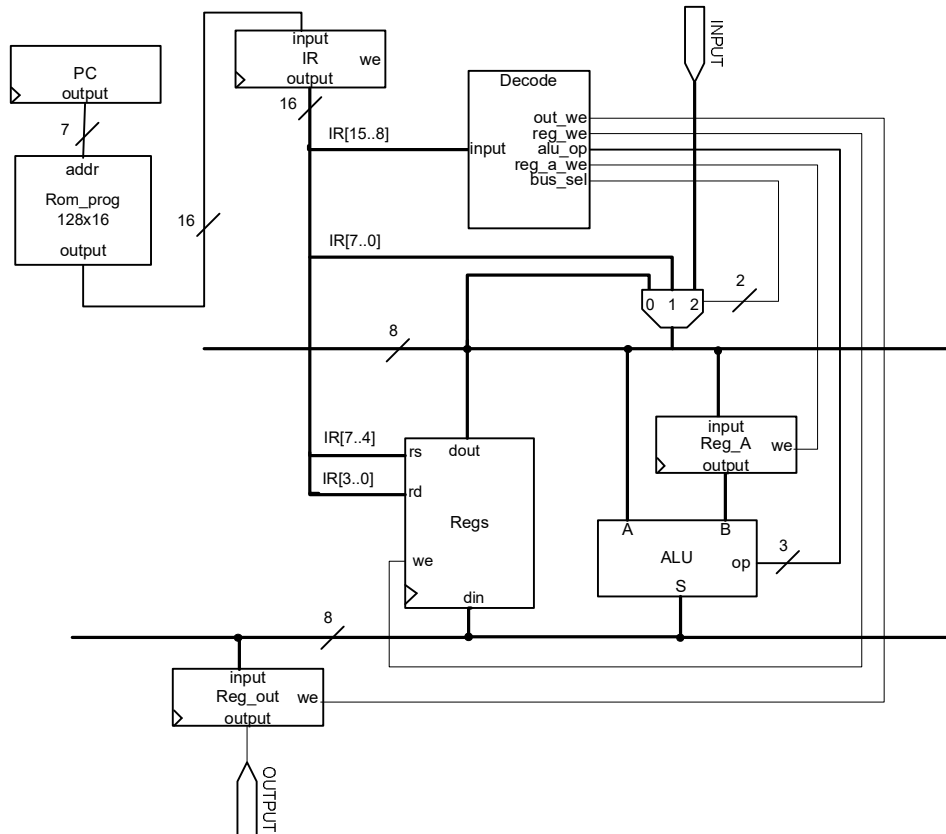


Trabajo Práctico Especial

En esta práctica estudiaremos el siguiente procesador:



Donde:

- **Memoria de instrucciones (ROM):** almacena el programa instrucción por instrucción
- **Contador de programa (PC):** dirección de instrucción a ejecutar
- **Registro de instrucción (IR):** almacena la instrucción actual a ejecutar
- **Unidad de decodificación (Decode):** determina las señales de control que deben activarse para ejecutar la instrucción.
- **Banco de registros (Regs):** compuesto por 16 registros de 8 bits
- **Registro acumulador (Reg_a):** permite almacenar temporalmente el operando B de la ALU
- **Unidad aritmético lógica (ALU)**
- **Registro de salida (Reg_out)**

Buses

Los buses de datos tienen un ancho de 8 bits, mientras que el de instrucción (IR) posee un ancho de 16 bits. En el esquema presentado se omiten las interconexiones de reloj y reset, para simplificar el diseño.

Instrucciones:

El registro de instrucción (IR) contiene la instrucción a ejecutar en el ciclo actual de operación y se divide en los siguientes campos:

Código de instrucción	rd	rs
8 bits	4 bits	4 bits

Código de instrucción	immediate
8 bits	8 bits

El procesador debe soportar las siguientes instrucciones con sus correspondientes códigos de operación.

IN rd

Código de instrucción: 0x01

Descripción: Reg[rd] = IN

OUT rs

Código de instrucción: 0x02

Descripción: Reg_out = Reg[rs]

MOV rd, rs

Código de instrucción: 0x03

Descripción: Reg[rd] = Reg[rs]

LDA rs

Código de instrucción: 0x04

Descripción: Reg_A = Reg[rs]

LDI immediate

Código de instrucción: 0x05

Descripción: Reg_A = immediate

ADD rd, rs

Código de instrucción: 0x10

Descripción: Reg[rd] = Reg[rs] + Reg_A

SUB rd, rs

Código de instrucción: 0x11

Descripción: Reg[rd] = Reg[rs] - Reg_A

SHL rd, rs

Código de instrucción: 0x20

Descripción: Reg[rd] = Reg[rs] << 1

SHR rd, rs

Código de instrucción: 0x21

Descripción: Reg[rd] = Reg[rs] >> 1

AND rd, rs

Código de instrucción: 0x12

Descripción: Reg[rd] = Reg[rs] and Reg_A

OR rd, rs

Código de instrucción: 0x13

Descripción: Reg[rd] = Reg[rs] or Reg_A

XOR rd, rs

Código de instrucción: 0x14

Descripción: Reg[rd] = Reg[rs] xor Reg_A

Unidad Aritmético Lógica (ALU)

La unidad aritmético lógica (ALU) opera con datos de 8 bits. Sus operaciones aritmético/lógicas básicas son suma, resta, and, or y xor. En el caso de introducir un código de operación (*op*) que no corresponda con los prefijados en la tabla, la señal de salida *S* se pondrá a 0.

El componente tiene como entradas a los operandos *a[7..0]*, *b[7..0]* y el selector de operación *op[2..0]*. La salida es *s[7..0]*.

Los códigos de operación se corresponderán con la siguiente tabla:

<i>op</i> [3..0]	Operación
000	<i>a</i>
010	<i>a + b</i>
011	<i>a – b</i>
100	<i>a and b</i>
101	<i>a or b</i>
110	<i>a xor b</i>

Banco de registros

Este banco de registros se compone de 16 registros de 8 bits. Las entradas del circuito están formadas, por una señal de reloj (*clk*), una señal de puesta a cero o reset (*rst*), una dirección de lectura (*rs*), una dirección de escritura (*rd*), el dato a escribir (*in*) y una habilitación de escritura (*we*). La salida está formada por el valor leído (*out*). La lectura se realiza en forma asíncrona.

De esta manera el comportamiento del circuito es el siguiente:

Lectura

$$out = Reg[rs]$$

Escritura

En flanco ascendente de reloj, si *we* = 1

$$Reg[Rd] = in$$

Realice

1. Complete la tabla e implemente la *Decode* (unidad de decodificación). Esta unidad permite la activación de las señales de control a partir del valor del código de instrucción almacenado en *IR*. Se recomienda realizar la minimización de las funciones.

Instrucción	<i>bus_sel</i>	<i>alu_op</i>	<i>reg_a_we</i>	<i>out_we</i>	<i>reg_we</i>
<i>IN</i> (0x01)	10	000	0	0	1
<i>OUT</i> (0x02)					
<i>MOV</i> (0x03)					
<i>LDA</i> (0x04)	00	000	1	0	0
<i>LDI</i> (0x05)					
<i>ADD</i> (0x10)					
<i>SUB</i> (0x11)	00	011	0	0	1
<i>AND</i> (0x12)					
<i>OR</i> (0x13)					
<i>XOR</i> (0x14)					
<i>SHL</i> (0x20)					
<i>SHR</i> (0x21)					

2. Implemente en VHDL a nivel de abstracción RTL, la Unidad de Decodificación, **utilizando sentencias concurrentes**. Mencione si el circuito implementado es secuencial o combinacional, justificando la respuesta.
3. Considerando que se cuenta con un banco de registros, *regs*, como el descrito en la siguiente entidad:

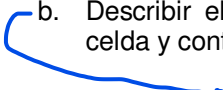
```
entity regs is
  Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        we : in  STD_LOGIC;
        rd : in  STD_LOGIC_VECTOR (2 downto 0);
        rs : in  STD_LOGIC_VECTOR (2 downto 0);
        din : in  STD_LOGIC_VECTOR (7 downto 0);
        dout : out STD_LOGIC_VECTOR (7 downto 0));
end regs;
```

Utilizando el componente *regs*, describa en VHDL el código que debe incluirse en el diseño del procesador para la implementación del banco de registro requerido. Nota: Se adjunta la versión completa del banco de registro, *regs*.

4. Determine la frecuencia de operación (frecuencia de reloj) óptima en que puede trabajar el procesador. Considere los siguientes tiempos
 - Tiempo de ROM: $T_{ROM} = 5 \text{ ns}$
 - Tiempo Multiplexores de 2 entradas de 32 bits: $T_{Mux2_32} = 2 \text{ ns}$;
 - Tiempo de demultiplexor de dos salidas de un bit: $T_{Dmux2_1} = 2 \text{ ns}$;
 - Tiempo de Unidad de Decodificación: $T_{Decode} = 4 \text{ ns}$
 - Tiempo de suma (usado en el *PC*): $T_{Add} = 5 \text{ ns}$
 - Tiempo de ALU: $T_{ALU} = 12 \text{ ns}$
 - Tiempo de registros: tiempo de set, $t_s = 0.2 \text{ ns}$; tiempo de propagación, $t_p = 0.8 \text{ ns}$; y tiempo del holding, $t_h = 0.02 \text{ ns}$.
5. Dado el siguiente programa, y considerando que todos los registros del procesador se encuentran inicializados en cero:

```
ldi 3
add r1, r0
ldi 5
add r2, r1
sub r3, r2
and r4, r1
xor r5, r1
or r7, r1
mov r14, r3
out r2
out r3
out r4
out r5
out r7
out r14
```

- a. Determine para cada ciclo de reloj, cuales son los registros que se actualizan, y el valor. Solo se debe especificar los registros correspondientes a los datos.
- b. Describir el contenido ROM para la ejecución de este programa, indicando celda y contenido.

 escribir el código en hexa de cada instrucción en una ROM