Guia practica 3 - Verilog Risc-V

Importante: De cada circuito es necesario diseñar su correspondiente banco de pruebas para detectar y corregir errores qué de otro modo serán difíciles de identificar en etapas posteriores de su aplicación. El banco de pruebas debe crearse y ejecutarse antes de pasar al siguiente desarrollo de circuito.

Parte 1:

Habiendo diseñado el rv32i, lo siguiente es realizar un prueba de funcionamiento de las instrucciones. Para eso tomamos un código de ejemplo y previo realizar algunas modificaciones particulares para nuestra implementación, podremos verificar en algun simulador su funcionamiento.

1. Podemos escribir el siguiente código en RARS.

```
#   Código de prueba:
#  Para probar este código en RARS,
#  deberá configurar la memoria como: Compact Data at address 0

#   Inicializo tres variables
        addi s0, zero, 3                    # a = 3
        addi s1, zero, 1                    # b = 1
        addi s2, zero, 16                   # cte = 16

#   operaciones lógicas aritméticas y slt
        or  t0, s0, s1                          # c = 3
        and t1, s0, s1                          # d = 1
        add t2, s0, s1                          # e = 4
        sub t3, s0, s1                          # f = 2
        sub t4, s1, s0                          # g = 0xfffffffe = -2
        slt t5, s0, s1                          # h = 0
        slt t6, s1, s0                          # i = 1
        slt t6, s1, t4,                     # j = 0

#   un while usando beq y j
#   Inicializo 3 variables
# s2 = 0x10 cte para comparar (16)
        addi t0, zero, 1                    # var = 1, variable de trabajo
        addi t1, zero, 0                    # cuenta = 0, un contador
while:  beq  t0, s2, sal1       # si var == cte, sale del while
        add  t0, t0, t0                     # var = var + var
        addi t1, t1, 1                      # cuenta = cuenta + 1
        j while
sal1:
#   Debió quedar var en 0x10 y cuenta en 4

#   un loop.
#   Inicializo 3 variables
# $t0 = i, $s1 = var
```

```
            add  s1, zero, zero                 # var = 0, $s0
            #add $s0, $0, 3                     # cte = 3, $s1
            addi t0, zero, 0                    # indice = 0, $t0
            addi t1, zero, 10               # veces = 10, $t1
for:    beq  t0, t1, sal2               # if indice == veces, branch to done
            add  s1, s1, s0                     # var = var + cte
            addi t0, t0, 1                      # incremento indice
            j for
sal2:
#    Debió quedar var en 30 (0x1e) e incremento en 10 (0xa)

#     almacenamiento (escritura) sw
            sw   s0, 0(zero)                    # guarda $s0 en registro 0
            sw   s1, 4(zero)                    # guarda $s1 en registro 4
            sw   s2, 8(zero)                    # guarda $s2 en registro 8
#    carga (lectura) lw
            lw   t0, 0(zero)                    # lee registro 0 en $t0
            lw   t1, 4(zero)                    # lee registro 4 en $t1
            lw   t2, 8(zero)                    # lee registro 8 en $t2
```

Con este código ensamblado debemos hacer ahora un **Dump Memory**, seleccionando el Dump Format como: **Hexadecimal text**, y guardando el archivo como .hex

La idea de haber usado *Compact Data at address 0* es en razón de poder usar las instrucciones *lw* y *sw* desde un puntero a la memoria de datos en la posición 0x00000000 como en nuestra implementación.

2. Por otra parte, ahora hay qué modificar las líneas de código referidas a direcciones de la memoria de instrucciones. Ya qué en el RARS, la dirección del segmento de .text apunta a 0x00003000 y en nuestra implementación el PC comienza en 0x00000000.
Concretamente:

Línea	Código	Cod máquina	Etiqueta/Dirección	Cod modificado
14	*beq t0, s2, sal1*	0x01228463	sal1 / 0x..3044	0x01228863
17	*jal zero, while*	0xffbff06f	while / 0x..3034	0xff5ff06f
21	*beq t0, t1, sal2*	0x00628463	sal2 / 0x..3060	0x00628863
24	*jal zero, for*	0xffbff06f	for / 0x..3050	0xff5ff06f

3. Con este archivo .hex modificado procedemos a copiar su contenido y pegarlo en el arreglo de la memoria de instrucciones.
Debería quedar algo así:...

```
module rom (
        input [4:0] address,            // Entrada de dirección de 5 bits
        output reg [31:0] data          // Salida de datos de 32 bits
```

```
);

// Declaración de contenido de la memoria ROM
reg [31:0] memory [0:31];

// Inicialización de la memoria ROM
initial begin
        memory[0] = 32'h00300413;  // Dirección 0
        memory[1] = 32'h00100493;  // Dirección 1
        memory[2] = 32'h01000913;  // Dirección 2
        memory[3] = 32'h009462b3;  // Dirección 3
        memory[4] = 32'h00947333;  // Dirección 4
        memory[5] = 32'h009403b3;  // Dirección 5
        memory[6] = 32'h40940e33;  // Dirección 6
        memory[7] = 32'h40848eb3;  // Dirección 7
        // seguir completando …
end

// Proceso de lectura de la ROM
always @ (address) begin
          data <= memory[address]; // Leer datos de la dirección de entrada
end
endmodule
```

Parte 2:

4. Con la memoria de instrucciones cargada con el código, deberá simular el comportamiento del rv32i y verificar qué en los registros del banco de registros se sucedan los resultados esperados de cada instrucción.