Práctica 4
Procesador: arquitectura, camino de datos y control

Nombre y Apellidos	Victor Correal Ramos
Nombre y Apellidos	Carlos Rodríguez Martínez

Número de grupo de laboratorio	1

Preguntas

1 Traduzca las siguientes instrucciones RISC-V a lenguaje ensamblador:

lenguaje máquina	lenguaje ensamblador
FF85AF03	slti rd, rs1, imm rd=0x1E rs=0x0B imm=0xFF8
00C735B3	sltu rd, rs1, rs1 rd=0x0B rs1=0x0E rs2=0x0C
10000297	auipc rd, imm rd=0x05 imm=0x10000

lenguaje máquina	lenguaje ensamblador
00E780A3	sb rs2, offset(rs1) rs2=0x0E rs1=0x0F offset=0x1
F7DFF0EF	jal rd, imm rd=0x1 imm=0xF7DFF
01D09463	bne rs1, rs2, imm rs1=0x1 rs2=0x1D imm=0x8

2 Un tipo numérico de datos en un lenguaje de alto nivel tiene un rango de valores limitado. En consecuencia, al efectuar operaciones algebraicas es posible que el resultado no se pueda representar (desbordamiento).

Suponga la operación de suma de los números naturales x e y, con un rango de valores 0 <= x, y < M, donde M-1 es el máximo valor que se puede representar. Por tanto, la operación x + y produce debordamiento cuando x > (M-1) - y.

Escriba una secuencia de instrucciones en lenguaje C que detecte desbordamiento sin efectuar la suma. Suponga que el valor máximo de un número natural está especificado por la constante MAX. En las operaciones que se especifiquen no debe producirse desbordamiento en ningún caso.

secuencia de instrucciones C

```
int x, y;
int desb; // 1 indica desbordamiento

desb = ((MAX^y)<x);
```

Suponga que los operandos se representan con vectores de 32 bits (M = MAX+1 = 2^{32}). Escriba una secuencia de instrucciones RISC-V en lenguaje ensamblador que detecte desbordamiento sin efectuar la suma. Tenga en cuenta que las operaciones de la secuencia no deben producir desbordamiento en ningún caso. El número de instrucciones debe ser el mínimo. Suponga que los operandos x e y están almacenados en los registros x10 y x11 respectivamente. El resultado se almacena en el registro x9 (el valor 1 indica desbordamiento).

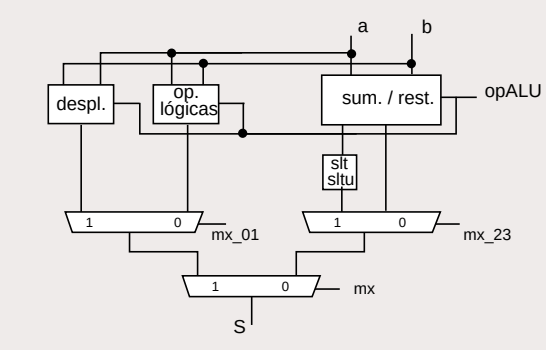
secuencia de instrucciones ensamblador

```
li x12, 0xFFFFFFFF # MAX
xor x9, x12, x11     # MAX^y
sltu x9, x9, x10     # (MAX^y)<x
```

3 Suponga que el procesador interpreta una instrucción de secuenciamiento condicional que cumple la condición. Indique las instrucciones de secuenciamiento que pueden generar los valores de salida del módulo EVAL (“Secuenciamiento condicional relativo“ en la página 288). Marque con X cualquier combinación que no se pueda producir.

ig	me	meu	instruccion secuenciamiento condicional
0	0	0	bne, bge, bgeu
0	0	1	bne, bge, bgeu, bltu
0	1	0	bne, bge, bgeu, blt
0	1	1	bne, bge, bgeu, bltu, blt
1	0	0	beq, bge, bgeu
1	0	1	beq, bge, bltu
1	1	0	beq, blt, bgeu
1	1	1	beq, blt, bltu

4 En la figura se muestran las 3 unidades funcionales de la ALU (página 292) y el árbol de multiplexores de selección: a) desplazamiento lógico o aritmético (a la derecha o a la izquierda), b) operación lógica, y c) sumador algebraico y comparador de menor (enteros o naturales). El módulo slt/sltu formatea (añade ceros a la izquierda) la salida de condición del sumador.

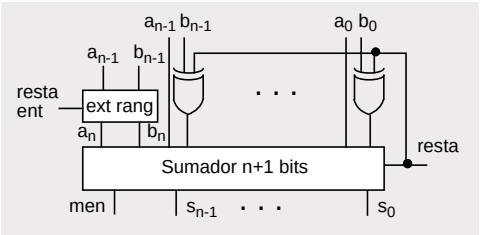


La ubicación de los ficheros relacionados con este proyecto (ALU.qpf) se indica en la página 295. Las dos primeras unidades funcionales y el formateador ya están diseñados (ficheros despla.vhd, logica.vhd y slt.vhd respectivamente). El fichero ALU.vhd contiene la descripción estructural de la ALU, excepto la selección de las salidas de las 3 unidades y el formateador. Esta seleccción se efectua mediante 2 niveles de multiplexores.

Deduzca las expresiones lógicas de las 3 señales de selección de los multiplexores en función de la señal de control opALU (“Señales de control de la ALU“ en la página 329). Inclúyalas en cuerpo de la arquitectura.

nivel 1	mx_01	opALU<2> == 0 | opALU<2:0>== 101
	mx_23	opALU<1> & 0x1
nivel 2	mx	mx_01 == 1 & opALU<2> !=0

El fichero sumalg.vhd contiene la interface del sumador algebraico y comparación de menor. Se utiliza un sumador de vectores de n+1 bits, puertas xor y lógica para extender el rango de los vectores a sumar. La salida men se activa cuando se cumple la condición a<b. Recuerde que los operandos se pueden interpretar como enteros o naturales. Esta salida se formatea (añadiendo ceros a la izquierda) en la unidad slt/sltu, que también está diseñada (fichero slt.vhd).



Analice el fichero sumalg.vhd. Deduzca las sentencias de asignación concurrente de las señales resta y ent (en funcion de opALU) y del bit más significativo de los vectores de entrada del sumador de n+1 bits (an, bn). Escriba las sentencias de asignación de las salidas de la unidad (en función del vector resultado de la suma).

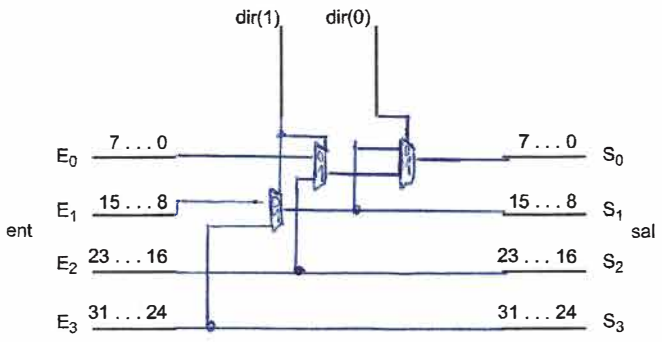
	resta	'0' when opALU = ALU_ADD else '1'	an	'0' when ent = '0' else '1' when a(tam_dat -1) = '1' and ent = '1'
	ent	'0' when opALU = ALU_ADD or ALU_SLTU else 1	bn	'0' when ent = '0' else '1' when b(tam_dat -1) = '1' and ent = '1'
salidas	men	suma (tam_dat)		
	s	suma(tam_dat-1 downto 0)		

Compruebe el diseño efectuado. El programa de prueba suministrado (prueba_ALU.vhd) compara, para un conjunto reducido de vectores, las salidas de la alu original y la diseñada.

5 El circuito FMTL (fichero FMTL.vhd) formatea el dato leído de la memoria que se utiliza para actualizar el banco de registros ("Segundo nivel: organización de la memoria de datos" en la página 295).

Analice el componente "alinear" y muestre un esquema del circuito. Como ayuda, utilice el visor RTL de quartus. Rellene la tabla que relaciona los datos de entrada y salida del módulo en función de los dos bits menos significativos de la dirección. Utilice la nomenclatura E_i y S_i para indicar el byte i del dato de entrada y de salida.

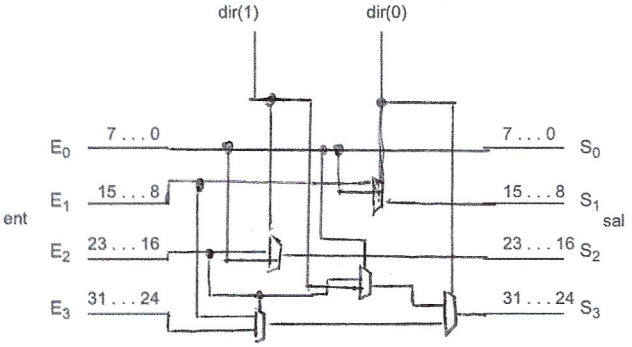
dir	sal			
1..0	S_3	S_2	S_1	S_0
0 0	E3	E2	E1	E_0
0 1	E3	E2	E1	E_1
1 0	E3	E2	E3	E_2
1 1	E3	E2	E3	E_3



6 El circuito FMTE formatea el dato leído del banco de registros para actualizar los bancos de la memoria de datos ("Segundo nivel: organización de la memoria de datos" en la página 295).

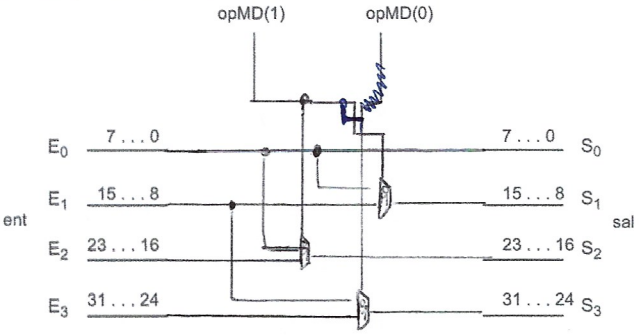
En el diseño base (fichero FMTE.vhd), el circuito consta de 2 módulos: a) alineamiento de datos (fichero alinearE.vhd), y b) selección de los bancos a actualizar (fichero sel_byte.vhd). El módulo alinearE utiliza los 2 bits menos significativos de la dirección.

Analice la descripción de la arquitectura del módulo alinearE y dibuje un esquema del circuito. Utilice el visor RTL de quartus. Rellene la tabla que relaciona los datos de entrada y salida del formateador en función de los dos bits menos significativos de la dirección.



dir	sal			
1..0	S_3	S_2	S_1	S_0
0 0	E_3	E_2	E_1	E_0
0 1	E_2	E_2	E_0	E_0
1 0	E_1	E_0	E_1	E_0
1 1	E_0	E_0	E_0	E_0

7 Proponga un diseño alternativo del módulo alinearE que utilice únicamente los 2 bits menos significativos de la señal de control opMD para formatear el dato de escritura a memoria ("Figura 4.107" en la página 332). La ubicación de los ficheros relacionados con este proyecto (ALINEAE.qpf) se indica en la página 298. En primer lugar rellene la tabla que relaciona los datos de entrada y salida del formateador en función de los dos bits menos significativos de la dirección. Minimice el número de niveles de selección y el número de multiplexores de 2 entradas.



opMD	sal			
1..0	S_3	S_2	S_1	S_0
0 0	E_1	E_0	E_0	E_0
0 1	E_1	E_0	E_0	E_0
1 0	E_3	E_2	E_1	E_0
1 1	X	X	X	X

Compruebe el diseño efectuado. Para ello modifique el programa de prueba.

8 Obtenga las siguientes métricas cuando el procesador base ejecuta el programa euclides: instrucciones ejecutadas, aritmético-lógicas, load, store, secuenciamiento condicional e incondicional ("Simulación de un programa concreto" en la página 306). Para ello, añada un proceso al programa de prueba (prueba_proc_MD_MI.vhd). Este proceso debe utilizar únicamente las señales de control opALU ("Figura 4.99" en la página 330), opMD ("Figura 4.107" en la página 332) y opSEC ("Figura 4.110" en la página 333) para determinar el tipo de instrucción interpretada en cada ciclo. Adjunte el código vhdl del proceso.

Instrucciones ejecutadas	103	Instrucciones aritmético-lógicas	65
Instruciones Load	2	Instrucciones Store	3
Instrucciones secuenciamiento condicional	25	Instrucciones secuenciamiento incondicional	7

```
--use work.cte_tipos_UF_pkg.all;
eje8: process is
	variable vInstruciones : integer := 0;
	variable vInstStore : integer := 0;
	variable vInstLoad : integer := 0;
	variable vInstAriLog : integer := 0;
	variable vInstSecCond : integer := 0;
	variable vInstSecIncond : integer := 0;

begin
	wait until reloj'event and reloj ='1';

	vInstruciones := vInstruciones + 1;
	if (s_opMD(4) = '1' and s_opMD(3) = '0') then
		vInstLoad := vInstLoad + 1;
	elsif (s_opMD(4) = '1' and s_opMD(3) = '1')then
		vInstStore := vInstStore + 1;

	elsif (s_opALU = ALU_ADD or s_opALU = ALU_SUB or s_opALU = ALU_SLL
	or s_opALU = ALU_SLT or s_opALU = ALU_SLTU or s_opALU = ALU_XOR or
s_opALU = ALU_SRL
	or s_opALU = ALU_SRA or s_opALU = ALU_OR or s_opALU = ALU_AND)
	then
		vInstAriLog := vInstAriLog + 1;

	elsif (s_opSEC = DECS_BEQ or s_opSEC = DECS_BNE or s_opSEC = DECS_BLT or
s_opSEC = DECS_BGE
			or s_opSEC = DECS_BLTU or s_opSEC = DECS_BGEU) then
		vInstSecCond := vInstSecCond  + 1;
	elsif (s_opSEC(3)) then
		vInstSecIncond := vInstSecIncond + 1;
	end if;

	report "instructions: " & integer'image(vInstruciones)
	& " arithmetic/logic: " & integer'image(vInstAriLog)
	& " Instruciones load : " & integer'image(vInstLoad)
	& " Instruciones store: " & integer'image(vInstStore)
	& " Instruciones sec Cond: " & integer'image(vInstSecCond)
	& " Instruciones sec Inco: " & integer'image(vInstSecIncond);

end process eje8;
```