David Cañadas López y Roger Ortega Castilló

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

LM	Ensamblador
FF85AF03	11111111100001011010111110000011 lw x30, -8(x11)
00C735B3	00000000110001110011010110110011 sltu x11, x14, x12
10000297	00010000000000000000010100101111 auipc x5, 65536
00E780A3	00000000111001111000000010100011 sb x14, 1(x15)
F7DFF0EF	11110111110111111111000011011111 jal x1, -132
01D09463	00000001110100001001010001100011 bne x1, x29, 8

Pregunta 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 desbordamiento 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.

Instrucciones en C
int x, y; int desb; // 1 indica desbordamiento

`if (x > (MAX-1) - y) {`
`    desb = 1;`
`} else {`
`    desb = 0;`
`}`

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).

David Cañadas López y Roger Ortega Castilló

Instrucciones en ensamblador

`addi x9, x0, 0xFFFFFFFF`
`sub x9, x9, x11`
`sltu x9, x9, x10`

Pregunta 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**	**Instrucción secuenciamiento condicional**
0	**0**	**0**	X (r1 != r2 & r1e >= r2e & r1n >= r2n)
0	**0**	**1**	bltu (r1 != r2 & r1e >= r2e & r1n < r2n) r1 = 1, r2 = -2
0	**1**	**0**	blt (r1 != r2 & r1e < r2e & r1n >= r2n)
0	**1**	**1**	blt (r1 != r2 & r1e < r2e & r1n < r2n)
1	**0**	**0**	beq (r1 == r2 & r1e >= r2e & r1n >= r2n) r1 = 2, r2 = 2
1	**0**	**1**	X (r1 == r2 & r1e >= r2e & r1n < r2n)
1	**1**	**0**	X (r1 == r2 & r1e < r2e & r1n >= r2n)
1	**1**	**1**	X (r1 == r2 & r1e < r2e & r1n < r2n)

Pregunta 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.

mx_01	`mx_01 <= not opALU(1) and opALU(0)`
mx_23	`mx_23 <= opALU(1)`
mx	`mx <= mx_01 or opALU(2)`

David Cañadas López y Roger Ortega Castilló

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	`opALU(3) xor opALU(1)`	**an**	`a(tam_dat-1)`
ent	`not opALU(3) and (opALU(1) xnor opALU(0))`	**bn**	`b(tam_dat-1)`
men	`‘1’ when a < b else ‘0’`		
s	`ss(tam_dat-1 downto 0)`		

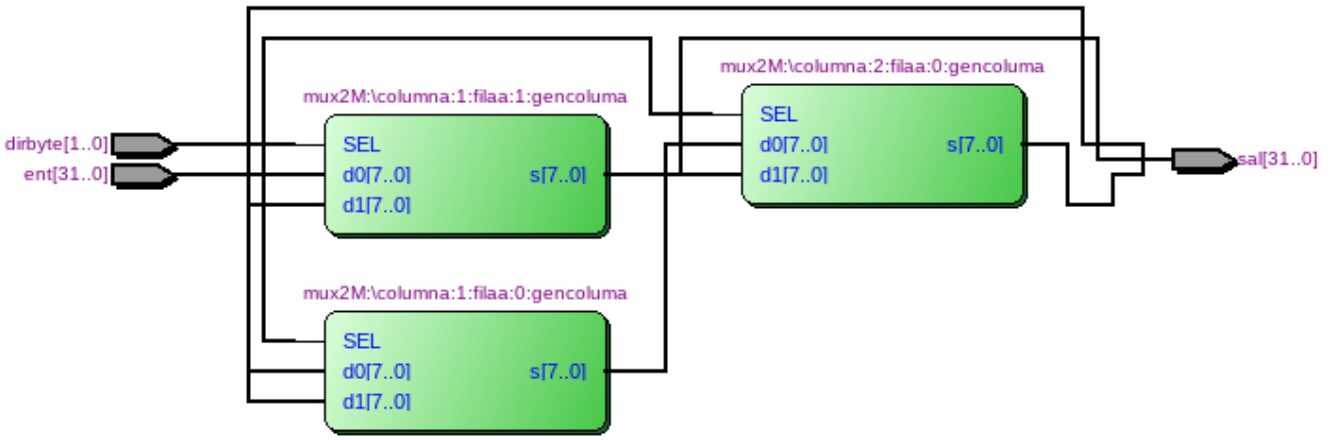
Compruebe el diseño efectuado. El programa de prueba suministrado (prueba_ALU.vhd) compara, para un conjunto reducido de vectores, las salidas de la ALU original y la diseñada.

Pregunta 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).

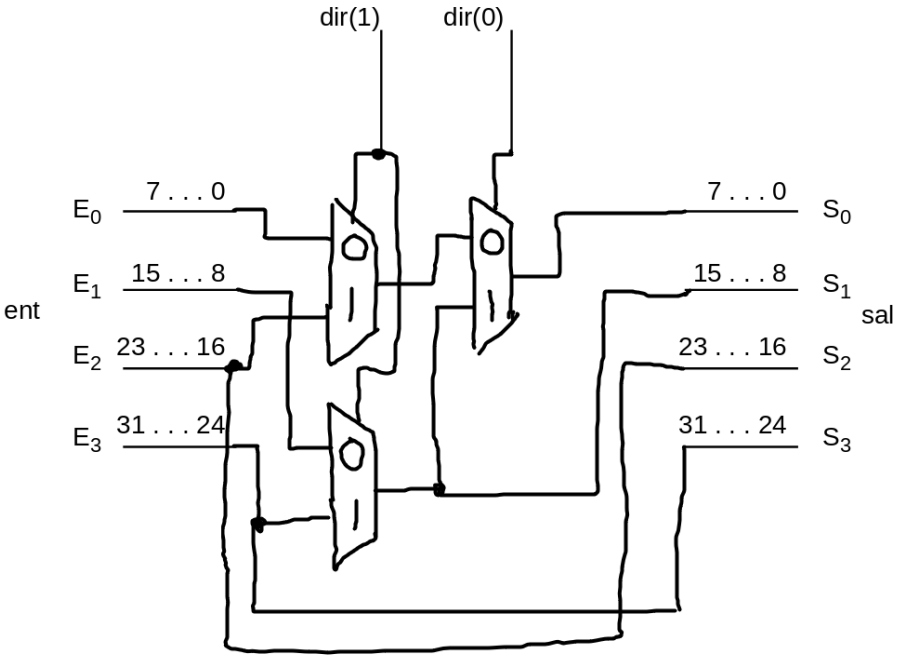
Analice el componente “alinear” y muestre un esquema del circuito. Como ayuda, utilice el visor RTL de quartus. Rellene la tabla que relaciona los datos de entrada y salida del módulo en función de los dos bits menos significativos de la dirección. Utilice la nomenclatura E i y S i para indicar el byte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

dir	**sal**			
1..0	**S3**	**S2**	**S1**	**S0**
0 0	E3	E2	E1	E0
0 1	E3	E2	E1	E1
1 0	E3	E2	E3	E2
1 1	E3	E2	E3	E3

En Quartus, la elaboración RTL produce el siguiente esquema:



Que se puede dibujar de la siguiente manera:

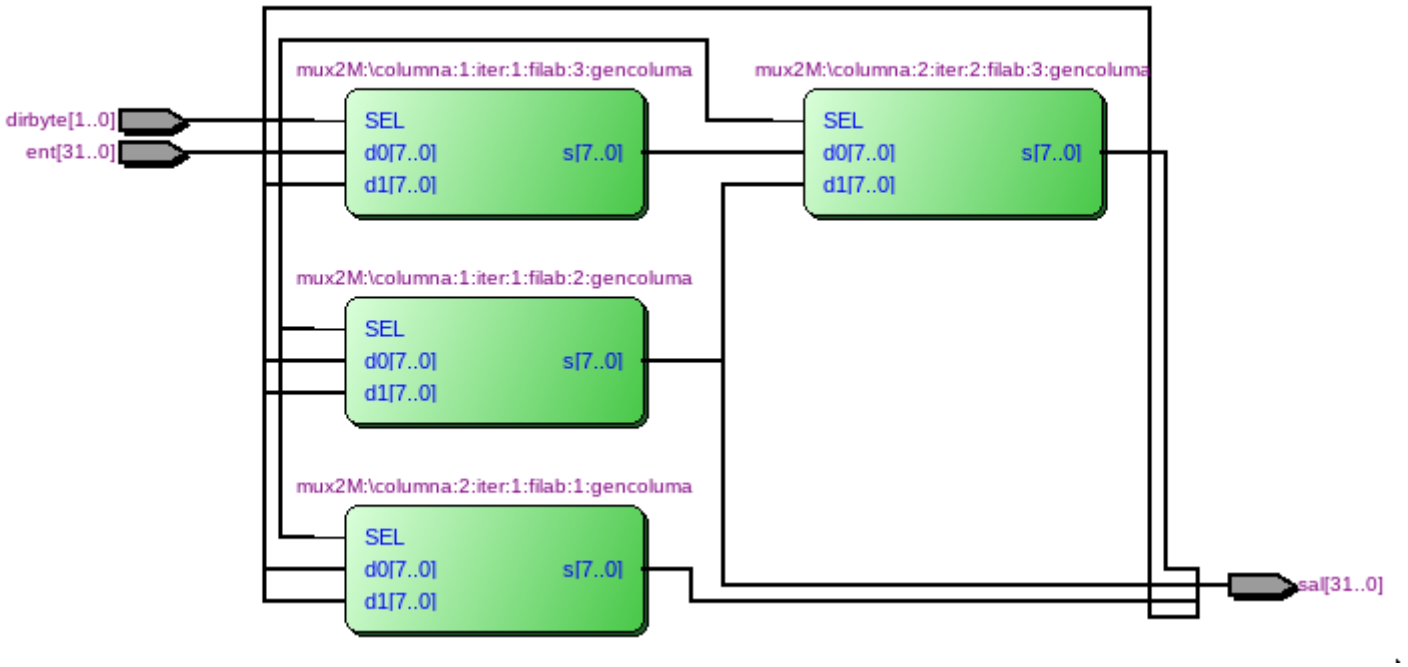


Pregunta 6: El circuito FMTE formatea el dato leído del banco de registros para actualizar los bancos de la memoria de datos (“Segundo nivel: organización de la memoria de datos“ en la página 295). En el diseño base (fichero FMTE.vhd), el circuito consta de 2 módulos: a) alineamiento de datos (fichero 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.

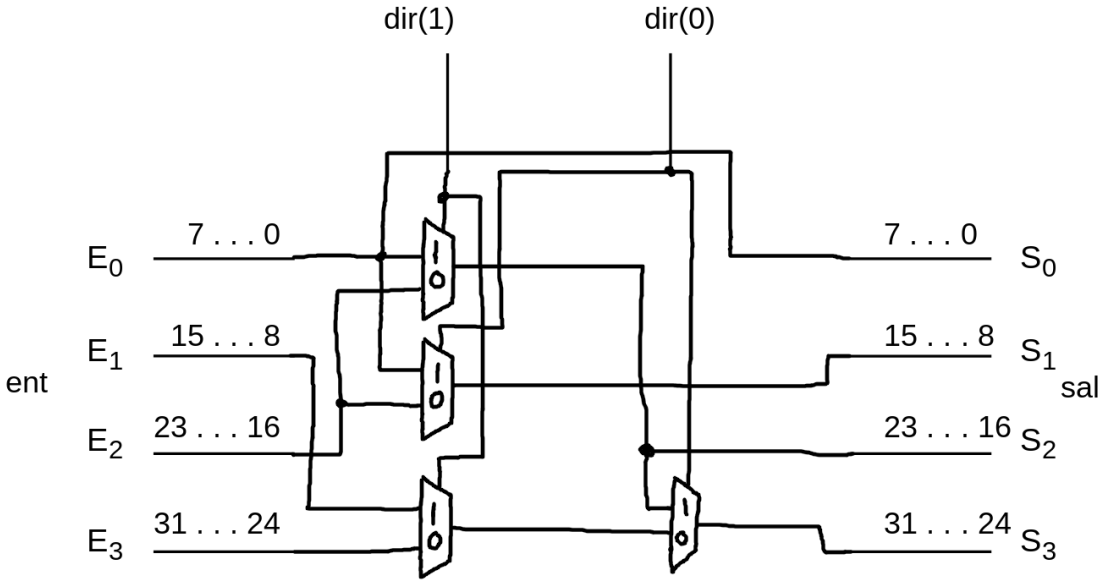
dir	**sal**			
1..0	**S3**	**S2**	**S1**	**S0**
0 0	E3	E2	E1	E0

0 1	E2	E2	E0	E0
1 0	E1	E0	E1	E0
1 1	E0	E0	E0	E0

En Quartus, la elaboración RTL produce el siguiente esquema:



Que se puede dibujar de la siguiente manera:

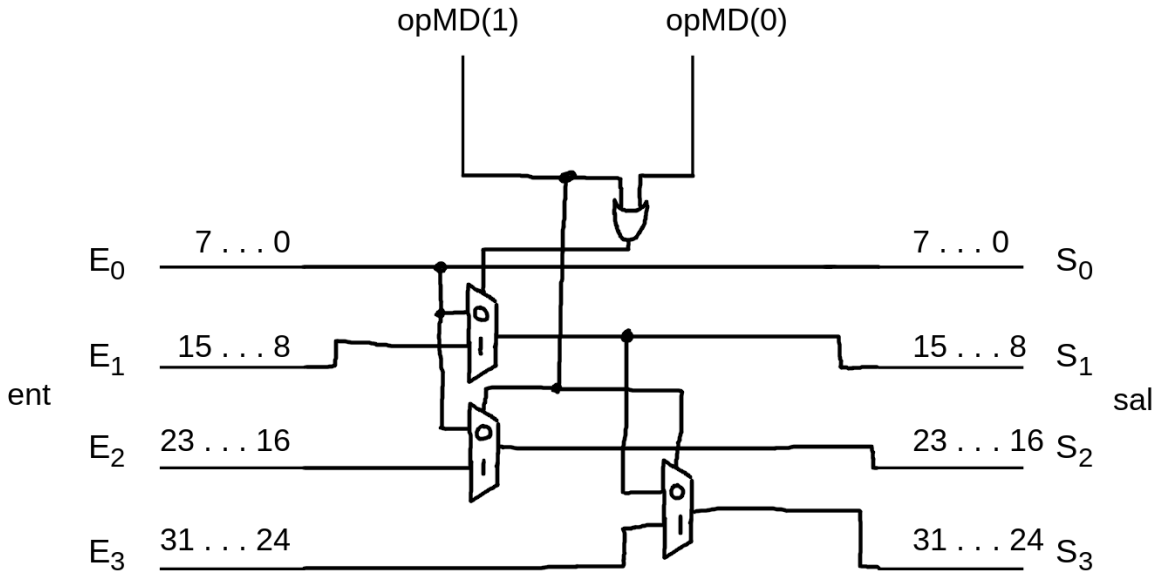


Pregunta 7: Proponga un diseño alternativo del módulo alinearE que utilice únicamente los 2 bits menos significativos de la señal de control opMD para formatear el dato de escritura a memoria (“Figura 4.107“ en la página 332). La ubicación de los ficheros relacionados con este proyecto (ALINEAE.qpf) 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. Compruebe el diseño efectuado. Para ello modifique el programa de prueba.

opMD	**sal**			
1..0	**S3**	**S2**	**S1**	**S0**
0 0	E0	E0	E0	E0
0 1	E1	E0	E1	E0
1 0	E3	E2	E1	E0
1 1	X	X	X	X

El diseño que proponemos se puede dibujar de la siguiente manera:



Pregunta 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	104
Instrucciones aritmético-lógicas	66
Instrucciones Load	2
Instrucciones Store	3
Instrucciones secuenciamiento condicional	26
Instrucciones secuenciamiento incondicional	7

El código vdhl del proceso añadido es:

```
contadores: process is
   variable i_totales: integer := 0;
   variable i_arilog: integer := 0;
   variable i_load: integer := 0;
   variable i_store: integer := 0;
   variable i_sec_cond: integer := 0;
   variable i_sec_icond: integer := 0;
begin
   wait until reloj'event and reloj ='1';
   i_totales := i_totales + 1;

   if s_opMD="10000" or s_opMD="10001" or s_opMD="10010" or s_opMD="10100" or
s_opMD="10101" then i_load := i_load + 1;
   elsif s_opMD="11000" or s_opMD="11001" or s_opMD="11010" then i_store := i_store
+ 1;
   elsif s_opALU="10000" or s_opALU="10001" or s_opALU="10010" or s_opALU="10011"
or s_opALU="10111" or s_opALU="11000" or s_opALU="11001" or s_opALU="11101" then
i_arilog := i_arilog + 1;
   elsif s_opSEC="1011" then i_sec_icond := i_sec_icond + 1;
   else i_sec_cond := i_sec_cond + 1;
   end if;

   report "Instrucciones totales: " & integer'image(i_totales);
   report "Aritmetico-logicas: " & integer'image(i_arilog);
   report "Loads: " & integer'image(i_load);
   report "Stores: " & integer'image(i_store);
   report "Sec. condicional: " & integer'image(i_sec_cond);
   report "Sec. incondicional: " & integer'image(i_sec_icond);
end process contadores;
```