Universitat Politècnica de Catalunya

Arquitectura de Computadores de Altas Prestaciones

**Práctica 4**

**Procesador: Arquitectura, camino de**

**datos y control**

Alumnos:

Manuel Velastegui

Carlos Andres Rodríguez Torres

**Grupo 5**

Cuatrimestre Primavera 2024-2025



**ÍNDICE**

[**1. Pregunta 1 3**](#_ucl316v6kc09)

[1.1 Tabla 1 3](#_ymhpiko7z0ar)

[1.2 Tabla 2 3](#_ee47gpqkoxia)

[1.3 Tabla 3 4](#_n3bn0jubdslt)

[1.5 Tabla 4 4](#_7p14d86gf4o1)

[1.5 Tabla 5 5](#_pzp5d2i5m8fa)

[1.6 Tabla 6 5](#_2f7v534it9zy)

[**2. Pregunta 2 5**](#_tun4wn8w50y1)

[2.1 Secuencia de Instrucciones C 6](#_kuvdgnd7jpt2)

[2.2 Secuencia de Instrucciones Ensamblador 6](#_qjnc7q9z3izz)

[**3. Pregunta 3 7**](#_ow5f5fi4v9lz)

[3.1 Secuenciamiento condicional relativo 7](#_kc3hracu5ik6)

[**4. Pregunta 4 7**](#_jqpanipbzz2h)

[4.1 Secuencias lógicas multiplexores 8](#_ky37df5a3mv4)

[4.1.1 Análisis expresión lógica mx\_01 9](#_ylluuhj9f69d)

[4.1.2 Análisis expresión lógica mx\_23 9](#_2jr550bny6he)

[4.1.3 Análisis expresión lógica mx 9](#_omr47nlvtfa6)

[4.2 Sentencia de asignaciones 10](#_c5xnpp21qe20)

[**5. Pregunta 5 11**](#_au1q582nhuz)

[5.1 Esquema del circuito 11](#_scrv14z9u0te)

[**6. Pregunta 6 12**](#_36knp09urhyj)

[6.1 Esquema del circuito 12](#_a3418ag7e5zh)

[**7. Pregunta 7 13**](#_fqnzzvku08p1)

[7.1 Tabla con Datos Entrada y Salida 14](#_h0zlp4o64g11)

[**8. Pregunta 8 15**](#_ylfvecw4fgzi)

[8.1 Tabla con Instrucciones 15](#_x0mpj04qgj4t)

[8.2 Esqueleto del Proceso 15](#_es190nymrag)

# 

# 1. Pregunta 1

Traduzca las siguientes instrucciones RISC-V a lenguaje ensamblador: Un campo

literal debe expresarse como entero en decimal, excepto en la instrucción “auipc” que

se expresa como entero en hexadecimal.

## 1.1 Tabla 1

En la Tabla 1, se muestra que la instrucción **FF85AF03** se traduce a lenguaje ensamblador cómo **lw R30, 4088 (R11)**. Donde se carga una palabra desde la dirección de memoria calculada como contenido de **R11** + **4088** y la almacena en el registro **R30.**

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FF85AF03 | F | | | | F | | | | 8 | | | | 5 | | | | A | | | | F | | | | 0 | | | | 3 | | | |
| FF85AF03 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Identificación campos | literal imm[11:0] | | | | | | | | | | | | rs1 | | | | | funct3 | | | rd | | | | | Op code | | | | | | |
| Campo literal | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| lw R30, 4088 (R11) | 4088 | | | | | | | | | | | | R11 | | | | | lw | | | R30 | | | | | LOAD | | | | | | |

*Tabla 1. Traducción instrucción FF85AF03*

## 1.2 Tabla 2

En la Tabla 2, se muestra que la instrucción **FF85AF03** se traduce a lenguaje ensamblador cómo **sltu R11, R12, R14.** La instrucción compara los valores sin signo de los registros R12 y R14. Si el valor en R12 es menor que el valor en R14, se coloca un 1 en R11; de lo contrario, se coloca un 0.

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 00C735B3 | 0 | | | | 0 | | | | C | | | | 7 | | | | 3 | | | | 5 | | | | B | | | | 3 | | | |
| 00C735B3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Identificación campos | 0 | 0 | 0 | 0 | 0 | 0 | 0 | rs2 | | | | | rs1 | | | | | funct3 | | | rd | | | | | Op code | | | | | | |
| sltu R11, R14, R12 | 0 | | | | | | | R12 | | | | | R14 | | | | | sltu | | | R11 | | | | | SLTU | | | | | | |

*Tabla 2. Traducción instrucción FF85AF03*

*00C735B3*

## 1.3 Tabla 3

La instrucción **10000297** se traduce a lenguaje ensamblador cómo **aupic R5 0x1000,** se toma el valor inmediato dado (en este caso, 0x10000), lo multiplica por 2^12 (o lo desplaza 12 bits hacia la izquierda, lo cual es equivalente a añadir doce ceros al final del número binario), y lo suma al valor actual del contador de programa (PC). El resultado se guarda en el registro R5.

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10000297 | 1 | | | | 0 | | | | 0 | | | | 0 | | | | 0 | | | | 2 | | | | 9 | | | | 7 | | | |
| 10000297 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| Identificación campos | imm[31:12] | | | | | | | | | | | | | | | | | | | | rd | | | | | Op code | | | | | | |
| Campo literal | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |
| auipc R5, 0x10000 | 0x10000 | | | | | | | | | | | | | | | | | | | | R05 | | | | | U-type | | | | | | |

*Tabla 2. Traducción instrucción 10000297*

## 1.5 Tabla 4

La instrucción **00E780A3** se traduce a lenguaje ensamblador cómo **SB R14, 0 (R15),** almacena el byte que está en el registro R14 en la dirección de memoria apuntada por el contenido de R15.

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 00E780A3 | 0 | | | | 0 | | | | E | | | | 7 | | | | 8 | | | | 0 | | | | A | | | | 3 | | | |
| 00E780A3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| Identificación campos | imm[11:5] | | | | | | | rs2 | | | | | rs1 | | | | | sb | | | imm[4:0] | | | | | SB | | | | | | |
| Campo literal | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |
| SB R14, 0 (R15) | 0 | | | | | | | R14 | | | | | R15 | | | | | sb | | | 0 | | | | | STORE | | | | | | |

*Tabla 4. Traducción instrucción 00E780A3*

## 1.5 Tabla 5

La instrucción **FEC79AE3** se traduce a lenguaje ensamblador cómo **bne R14,R12, 0x8149,** hace que el programa salte a la dirección calculada si los valores de R14 y R12 no son iguales.

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FEC79AE3 | F | | | | E | | | | C | | | | 7 | | | | 9 | | | | A | | | | E | | | | 3 | | | |
| FEC79AE3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| Identificación campos | imm[12|10:5] | | | | | | | rs2 | | | | | rs1 | | | | | BNE | | | imm[4:1|11] | | | | | Op code | | | | | | |
| Campo literal | 1 | 1 | 1 | 1 | 1 | 1 | 0 |  | | | | |  | | | | |  | | | 1 | 0 | 1 | 0 | 1 |  |  |  |  |  |  |  |
| bne R14,R12, 0x8149 | imm[12|10:5] | | | | | | | R12 | | | | | R15 | | | | | BNE | | | imm[4:1|11] | | | | | BRANCH | | | | | | |

*Tabla 5. Traducción instrucción FEC79AE3*

## 1.6 Tabla 6

La instrucción **FD1FF06F** se traduce a lenguaje ensamblador cómo **jal R0,0x1036799,** almacena la dirección de la siguiente instrucción en el registro proporcionado (en este caso, R0) para poder regresar más tarde al flujo de ejecución actual. Salta a la dirección que se calcula sumando el offset inmediato al PC actual.

|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FD1FF06F | F | | | | D | | | | 1 | | | | F | | | | F | | | | 0 | | | | 6 | | | | F | | | |
| FD1FF06F | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| Identificación campos | imm[20|10:1|11|19:12] | | | | | | | | | | | | | | | | | | | | rd | | | | | Op code | | | | | | |
| jal R0,0x1036799 | 0x1036799 | | | | | | | | | | | | | | | | | | | | R0 | | | | | JAL | | | | | | |

*Tabla 5. Traducción instrucción FD1FF06F*

# 2. 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

## 2.1 Secuencia de Instrucciones C

| Entradas (t<t0) | | | Entradas (t=t0) | | | Retardo(t1 – t0) |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | Cen | A | B | Cen | Ret1(ns) |
| 0000 | 0000 | 0 | 0000 | 0100 | 0 | 30ns |

*Tabla 6. Retardos aplicados*

## 2.2 Secuencia de Instrucciones Ensamblador

Suponga que los operandos se representan con vectores de 32 bits (M = MAX+1 = 232).

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 | Justificación |
| --- | --- |
| lui x12, 0xFFFFF # Cargar los 20 bits superiores de MAX en x12  ori x12, x12, 0xFFF # Completar los bits inferiores de MAX (esto resulta en x12 = 0xFFFFFFFF)  sub x9, x12, x10 # Restar x de MAX y almacenar el resultado en x9  sltu x9, x9, x11 # Establecer x9 a 1 si x9 es menor que y, lo que indica desbordamiento | |

*Tabla 7. Secuencia de instrucciones ensamblador*

El programador utiliza la instrucción lui (load upper immediate) para cargar los 20 bits más significativos del valor máximo representable en un registro de 32 bits (en este caso 0xFFFFF), y lo coloca en el registro x12. Esto es necesario porque en la arquitectura RISC-V no existe una instrucción inmediata que permita cargar directamente un valor completo de 32 bits en un registro; en cambio, debe hacerse en dos pasos.

# 3. 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 246). Marque con X cualquier combinación que no se pueda producir.

## 3.1 Secuenciamiento condicional relativo

Siguiendo el contenido Figura 4.111 de la pág 294 y la Figura 4.85 de la pág 282. Hemos determinado la siguiente tabla resumen:

| IG | bne | no igual |
| --- | --- | --- |
| Beq | Igual |
| ME | Blt | < |
| Bge | >= |
| MEU | Bgeu | >= u |
| bltu | <u |

*Tabla 8. Tipo de instrucción analizada por bit a analizar*

Y finalmente el siguiente resultado:

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

*Tabla 9. Resultados*

caso 1 0 1 → es igual, es menor en entero y mayor en natural

Para poder sacar la información de la tabla se han analizado las diferentes posibilidades y sabiendo que cada caso podía tener 3 instrucciones de secuenciamiento ya que hay 3 bits, de cada tipo de instrucción se ha buscado la combinación de 3 instrucciones que cumpliesen los 3 casos. Para los 3 últimos casos no existe combinación posible.

# 4. Pregunta 4

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 289). Inclúyalas en cuerpo de la arquitectura.

## 4.1 Secuencias lógicas multiplexores

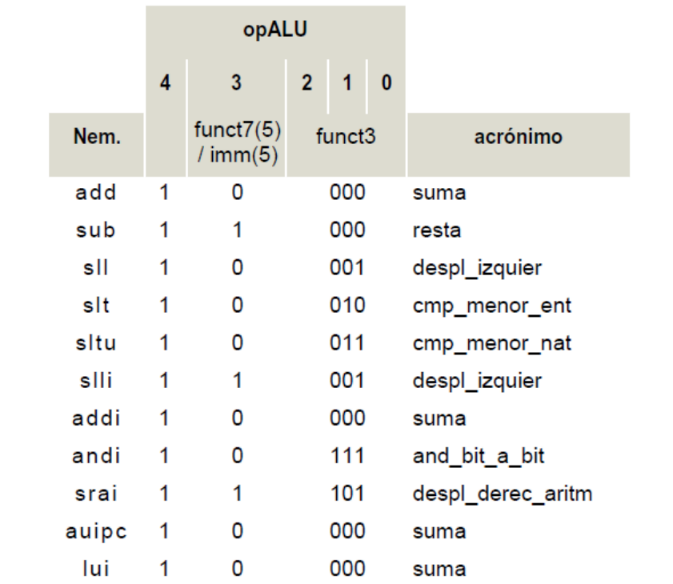
## 

*Figura 1. Secuencia de multiplexores*

| nivel 1 | mx\_01  mx\_23 | !opALU[3] and !opALU[1] and opALU[0] |
| --- | --- | --- |
| !opALU[2]and opALU[1] |
| nivel 2 | mx | OpALU[2] or (!opALU[1] and opALU[0]) |

*Tabla 10. Expresiones lógicas de las señales*

Para determinar los valores, se utilizaron los valores que se ven en la tabla de la página 290, donde a partir de la figura 8, utilizamos un excel para sacar de manera lógica y por medio de análisis de bits de cada opción las expresiones lógicas.



*Tabla 11 . Codificación de la señal opALU*

### 4.1.1 Análisis expresión lógica mx\_01

|  | **4** | **3** | **2** | **1** | **0** |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **sll** | 1 | 0 | 0 | 0 | 1 | DESP | mx01(1)  mx(1) |
| **slli** | 1 | 1 | 0 | 0 | 1 | DESP |
| **srai** | 1 | 1 | 1 | 0 | 1 | DESP |
| **andi** | 1 | 0 | 1 | 1 | 1 | AND | mx01(0)  mx(1) |

*Tabla 12 . Análisis señal lógica mx\_01*

En este caso, hemos de mirar que opALU[0] siempre sea 1 y también el contrario de opALU[1] (!opALU[1]), es decir, si el bit 1 de opALU es 0, corresponderá con una instrucción de desplazamiento; por tanto, el valor de mx01 será 1, correspondiendo al valor contrario de opALU[1].En el caso de que el bit 1 de opALU sea 1, corresponderá a una operación lógica y, por tanto, mx01 será 0.

### 4.1.2 Análisis expresión lógica mx\_23

|  | 4 | 3 | 2 | 1 | 0 |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slt | 1 | 0 | 0 | 1 | 0 | CMP | mx23(1)  mx(0) |
| Sltu | 1 | 0 | 0 | 1 | 1 | CMP |
| Sub | 1 | 1 | 0 | 0 | 0 | RESTA | mx23(0)  mx(0) |
| Add | 1 | 0 | 0 | 0 | 0 | SUMA |
| Addi | 1 | 0 | 0 | 0 | 0 | SUMA |
| auipc | 1 | 0 | 0 | 0 | 0 | SUMA |
| lui | 1 | 0 | 0 | 0 | 0 | SUMA |

*Tabla 13 . Análisis señal lógica mx\_23*

En este caso, si opALU[1], es decir, el bit 1 de opALU, es 0, corresponderá con una instrucción de suma o resta; por tanto, el valor de mx23 será 0, igual al valor de opALU[1].En el caso de que el bit 1 de opALU sea 1, corresponderá a una operación slt o stlu y, por tanto, mx23 será 1.

### 4.1.3 Análisis expresión lógica mx

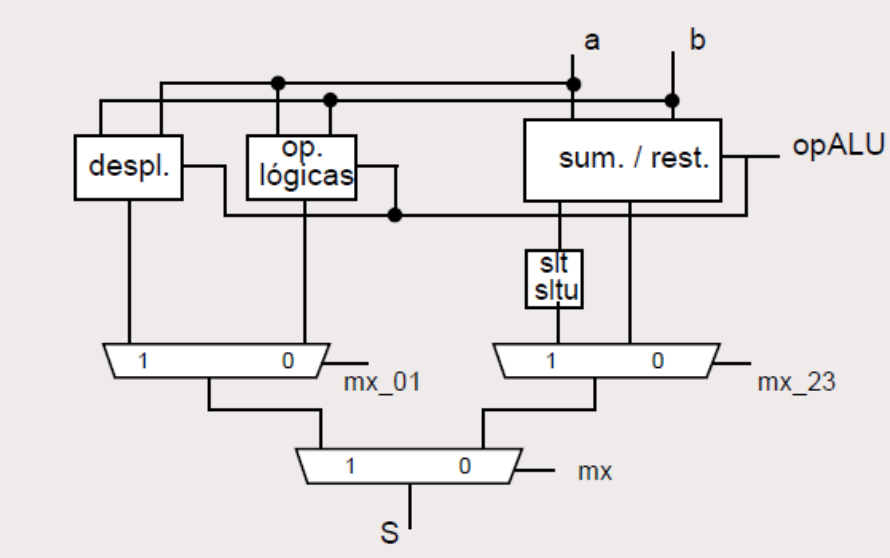
|  | 4 | 3 | 2 | 1 | 0 |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slt | 1 | 0 | 0 | 1 | 0 | CMP | Mx23(1)  Mx(0) |
| Sltu | 1 | 0 | 0 | 1 | 1 | CMP |
| Sub | 1 | 1 | 0 | 0 | 0 | RESTA | Mx23(0)  Mx(0) |
| Add | 1 | 0 | 0 | 0 | 0 | SUMA |
| Addi | 1 | 0 | 0 | 0 | 0 | SUMA |
| Auipc | 1 | 0 | 0 | 0 | 0 | SUMA |
| Lui | 1 | 0 | 0 | 0 | 0 | SUMA |
| Sll | 1 | 0 | 0 | 0 | 1 | DESP | Mx01(1)  Mx(1) |
| Slli | 1 | 1 | 0 | 0 | 1 | DESP |
| Srai | 1 | 1 | 1 | 0 | 1 | DESP |
| Andi | 1 | 0 | 1 | 1 | 1 | AND | Mx01(0)  Mx(1) |

*Tabla 14 . Análisis señal lógica mx*

Para este caso hemos usado una herramienta online nos ha dado y = B + C’D, sabiendo que la suma es una or y la multiplicación una and, hemos sacado la expresión, también teniendo en cuenta que el bit a era el 3 y el d el 0.

## 4.2 Sentencia de asignaciones

Analice el fichero sumalg.vhd. Deduzca las sentencias de asignación concurrente de las señales resta y ent (en función 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).



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).

|  | resta  ent | opALU[4] and opALU[0] | an  bn | ent and a(31) |
| --- | --- | --- | --- | --- |
| OpALU[0] | ent and b(31) |
| salidas | men  s | S(32) |  | |
| S[0..31] |

*Tabla 15. Expresiones lógicas sumador de n+1 bits*

Para deducir los datos de esta tabla simplemente se han seguido la tabla y con los valores de la tabla de opALU igual que en el ejercicio de antes se han deducido los valores.

# 5. Pregunta 5

El circuito FMTL (fichero FMTL.vhd) formatea el dato leído de la memoria que se utiliza

para actualizar el banco de registros.

Analice el componente “alinear” (FMTL/ALINEAR/CODIGO) y muestre un esquema del

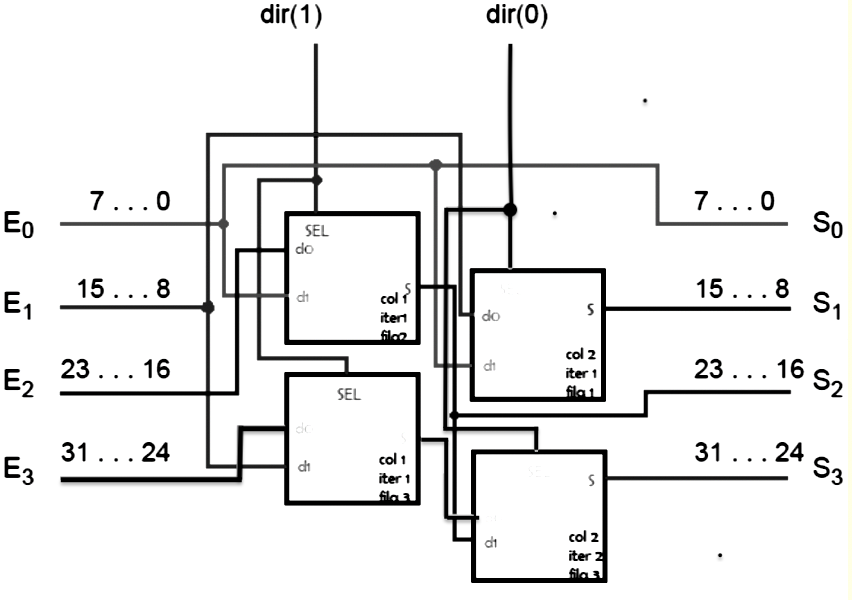
circuito. Como ayuda, utilice el visor RTL de quartus (elabore el proyecto entero, ENSAM-

BLADO). 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 Ei y Si

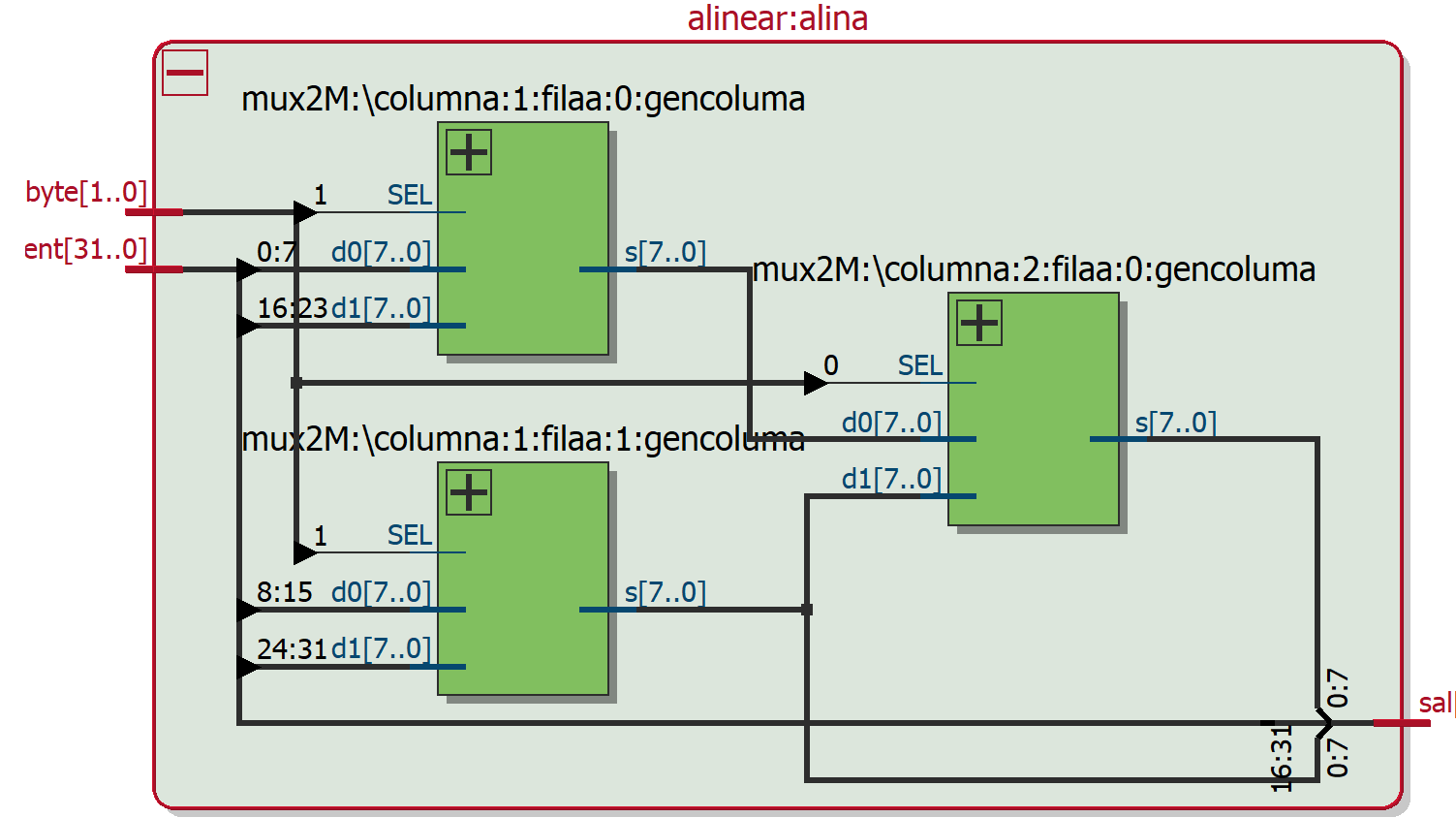
para indicar el byte i del dato de entrada y de salida.

## 5.1 Esquema del circuito

A partir del modelo en Quartus, se puede obtener el circuito y por tanto, seguir las salidas para completar la tabla.

| **Dirección 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 |

*Tabla 16. Direcciones y salidas*



*Figura 2. Direcciones y salidas*

# 6. Pregunta 6

El circuito FMTE formatea el dato leído del banco de registros para actualizar los

bancos de la memoria de datos.

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

(FMTE/ALINEARE/CODIGO) y dibuje un esquema del circuito. Utilice el visor RTL de

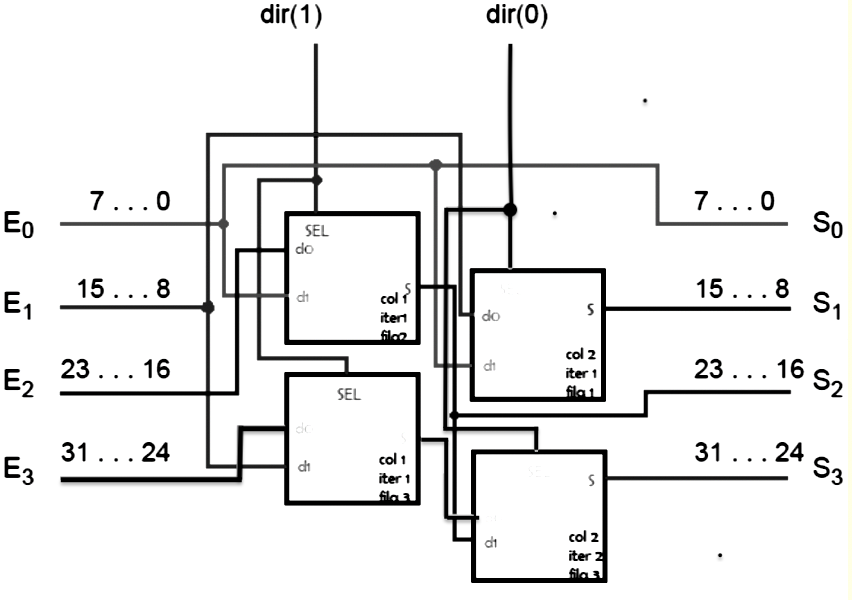
quartus (elabore el proyecto entero, ENSAMBLADO). Rellene la tabla que relaciona los

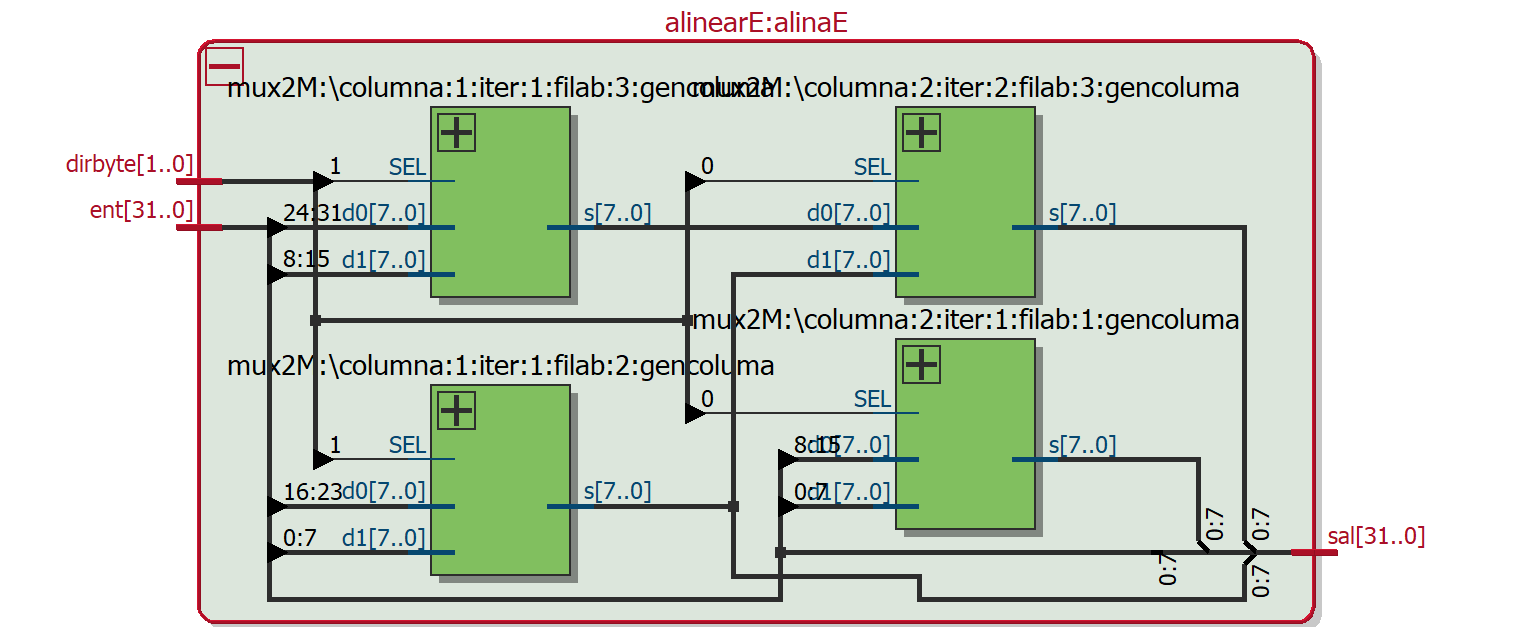
datos de entrada y salida del formateador en función de los dos bits menos significativos

de la dirección.

## 6.1 Esquema del circuito

| Dir 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 |





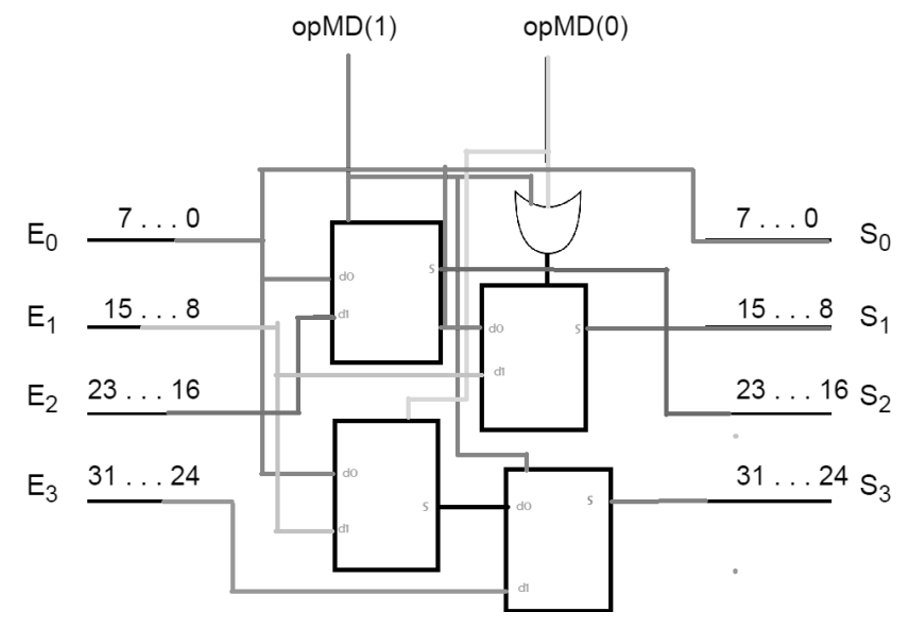
# 7. Pregunta 7

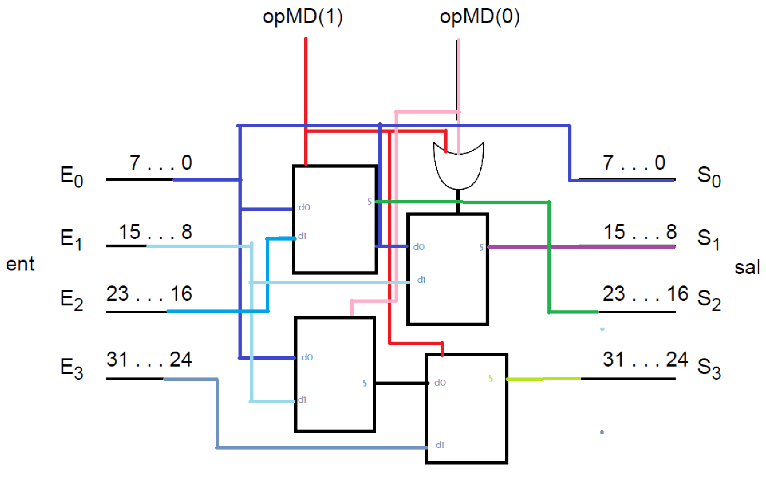
Proponga un diseño alternativo del módulo alinearE que utilice únicamente los 2 bits

menos significativos de la señal de control opMD para formatear el dato de escritura a

memoria. La ubicación de los ficheros relacionados con este proyecto (ALINEAE.qpf) se indica en la página 256. 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.

## 7.1 Tabla con Datos Entrada y Salida





*Figura 3. Propuesta diseño*

| opMD(1..0) | S[31..24] | S2[23..16] | S1[15..8] | S[7…0] |
| --- | --- | --- | --- | --- |
| 0 0 | E0[7..0] | E0[7..0] | E0[7..0] | E0[7..0] |
| 0 1 | E[15..8] | E0[7..0] | E0[7..0] | E0[7..0] |
| 1 0 | E[31..24] | E[23..16] | E[15..8] | E0[7..0] |
| 1 1 | X | X | X | X |

*Tabla 18. Direcciones y salidas*

# 8. Pregunta 8

Obtenga las siguientes métricas cuando el procesador base ejecuta el programa euclides: instrucciones ejecutadas, aritmético-lógicas, load, store, secuenciamiento condicional e incondicional.

Para ello, añada un proceso al programa de prueba. Este proceso debe utilizar únicamente las señales de control opALU , opMD y opSE para determinar el tipo de instrucción interpretada en cada ciclo. Recuerde la forma de referenciar objetos en un diseño jerárquico utilizando referencias jerárquicas.

## 8.1 Tabla con Instrucciones

| Instrucciones ejecutadas | 103 |
| --- | --- |
| Instrucciones Load | 2 |
| Instrucciones secuenciamiento condicional | 25 |

| Instrucciones aritmético-lógicas | 66 |
| --- | --- |
| Instrucciones Store | 3 |
| Instrucciones secuenciamiento incondicional | 7 |

*Tabla 19. Tabla con instrucciones*

## 8.2 Esqueleto del Proceso

estadistiques: process is

variable instrStore : integer := 0;

variable instrLoad : integer := 0;

variable instrAritmLog: integer := 0;

variable instrSCond : integer := 0;

variable instrSIncond : integer := 0;

begin

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

if (s\_opMD(4) = '1' and s\_opMD(3) = '0') then

instrLoad := instrLoad + 1;

elsif (s\_opMD(4) = '1' and s\_opMD(3) = '1') then

instrStore := instrStore + 1;

elsif (s\_opALU = ALU\_ADD or

s\_opALU = ALU\_SUB or

s\_opALU = ALU\_AND) then

instrAritmLog := instrAritmLog + 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

instrSCond := instrSCond + 1;

elsif (s\_opSEC(3)) then instrSIncond := instrSIncond + 1;

end if;

report "Instrucciones Aritmetico Logicas: " & integer'image(instrAritmLog)

& " Instrucciones Load: " & integer'image(instrLoad)

& " Instrucciones Store: " & integer'image(instrStore)

& " Instrucciones Secuenciamiento Condicional: " & integer'image(instrSCond)

& " Instrucciones Secuenciamiento Incondicional: " & integer'image(instrSIncond);

end process;

Resultado en console output → “103 0000009C 00100073 ebreak 00 0 0 10 00 1 00 00000 00000 0000 0 1 0 0 00000000 0 0 000000A0

# \*\* Note: Instrucciones Aritmetico Logicas: 66 Instrucciones Load: 2 Instrucciones Store: 3 Instrucciones Secuenciamiento Condicional: 25 Instrucciones Secuenciamiento Incondicional: 7”

## 