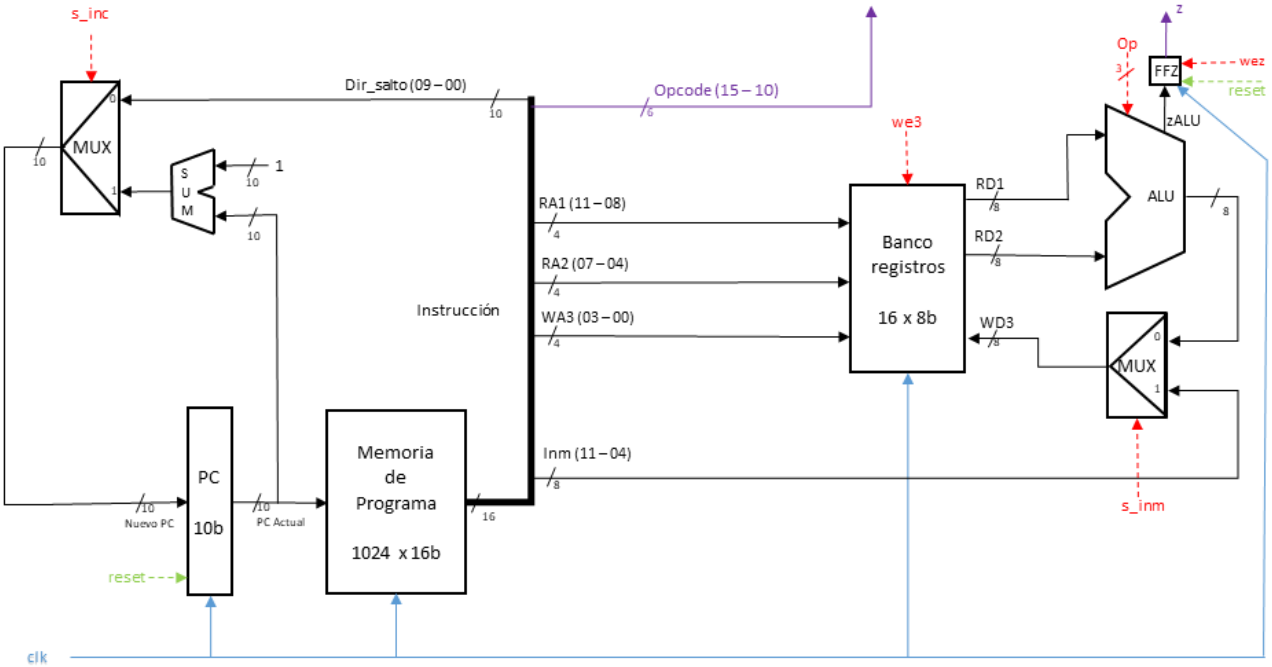
DESCRIPCIÓN DEL PROBLEMA

El objetivo de esta segunda sesión práctica previa es prepararnos para la práctica logrando una mejor comprensión sobre cómo funcionan el procesador y su unidad de control. Para ello, nos centraremos en un procesador muy simple de un solo ciclo. Para que un procesador pueda ejecutar instrucciones en un solo ciclo sin recurrir al paralelismo en su implementación debemos separar las memorias de instrucciones y de datos de forma que se pueda realizar el acceso a ambas dentro del mismo ciclo (al estilo de la arquitectura Harvard). En este ejemplo el procesador no va a tener una memoria de datos propiamente dicha, sino que operará con su banco de registros como memoria de datos. Esta estructura es típica de algunos microcontroladores, procesadores muy sencillos con una memoria de programa no volátil, diseñados para funcionar integrados en otro artefacto como una lavadora o un coche, por ejemplo.

Para analizar el funcionamiento del procesador, estudiaremos separadamente el camino de datos de la unidad de control que lo gobierna y los modelaremos por separado también. En la figura se han marcado en rojo y con línea discontinua las señales que provendrían de la Unidad de Control. Las instrucciones están codificadas en la memoria de programa, siendo cada una de 16 bits. El PC es de 10 bits, permitiendo direccionar un total de 1024 instrucciones en memoria de programa. Hay 16 registros de 8 bits en el banco de registros.



La unidad de control para este camino de datos tendría como entradas

- las señales comunes de reloj y reset
- los 6 bits más significativos de la instrucción (Opcode mayor posible)
- el valor del flag de cero para la ejecución de los saltos condicionales

y generaría las señales de control:

- señales de control de ambos multiplexores (s_inc y s_inm)
- la habilitación de escritura del banco de registros (we3)
- señales de selección de operación de la ALU (Op)
- la habilitación de escritura del flag de cero (wez)

La misión de la unidad de control será activar las señales de control de forma que se ejecute correctamente la instrucción determinada por los 6 bits (o menos) de Opcode.

La siguiente tabla describe tanto la codificación (tal como aparecería en memoria de programa) como el funcionamiento de cada una de las instrucciones. En la tabla se usan las abreviaturas siguientes- X: valores arbitrarios, C: bits de una constante, D: bits del destino de un salto, Op: señales de selección de operación de la ALU, R1: bits índice del registro primer operando, R2: igual para el registro segundo operando y Rd: igual que los casos anteriores pero para el registro destino.

Instrucción	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
No Operación (nop)	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X
No realiza ninguna acción, más allá de avanzar a la siguiente instrucción. Opcode de 6 bits (15-10). Esta instrucción no realiza ninguna acción visible al programador salvo pasar a la siguiente instrucción provocando que el nuevo PC sea el PC actual incrementado en 1.																
Carga inmediata (li)	0	0	0	1	C	C	C	C	C	C	C	C	Rd	Rd	Rd	Rd
Carga un valor inmediato en un registro. Opcode de 4 bits (15-12), constante inmediata de 8 bits (11-4) y campo de registro de destino de 4 bits (3-0) indicando el índice del registro destino (WA3) donde se escribirá la constante siempre que el multiplexor que provee el dato a escribir tenga la entrada s_inm a 1.																
Salto incondicional (j)	0	1	0	0	0	0	D	D	D	D	D	D	D	D	D	D
Continúa la ejecución del código en la dirección PC+Desplazamiento. Opcode de 6 bits (15-10) y los 10 bits restantes (9-0) serán el nuevo PC si el multiplexor que controla la entrada al PC tiene su entrada de selección s_inc a cero. En las instrucciones que no sean saltos, s_inc se pondrá a 1 provocando que el nuevo PC sea el PC actual incrementado en 1.																
Salto condicional si cero (jz)	0	1	0	0	0	1	D	D	D	D	D	D	D	D	D	D
Continúa la ejecución del código en la dirección PC+Desplazamiento si alguna operación anterior dio, como resultado, 0. Opcode de 6 bits (15-10) y los 10 bits restantes (9-0) serán el nuevo PC si el flag de cero vale 1 (z=1). En caso contrario (z=0), el nuevo PC será el PC actual incrementado en 1.																
Salto condicional si no cero (jnz)	0	1	0	0	1	0	D	D	D	D	D	D	D	D	D	D
Continúa la ejecución del código en la dirección PC+Desplazamiento si alguna operación anterior dio, como resultado, algo diferente a 0. Opcode de 6 bits (15-10) y los 10 bits restantes (9-0) serán el nuevo PC si el flag de cero vale 0 (z=0). En caso contrario (z=1), el nuevo PC será el PC actual incrementado en 1.																
Operación aritm./lógica (add, …)	1	Op	Op	Op	R1	R1	R1	R1	R2	R2	R2	R2	Rd	Rd	Rd	Rd
Realiza una operación aritmético/lógica entre dos registros y guarda el resultado en el registro Rd. Opcode de 4 bits (15-12, de los cuales 14-12 representan la señal de control Op que se enviará a la ALU), campo índice de primer registro operando de 4 bits (11-8, RA1), campo índice de segundo registro operando de 4 bits (7-4, RA2) y campo índice de registro de destino de 4 bits (3-0, WA3) donde se almacenará el resultado (siempre que el multiplexor tenga s_inm a cero). Estas instrucciones son las únicas que deben afectar al flag de cero z.																

TAREAS A REALIZAR: SIMULACIÓN DE UNIDAD DE CONTROL A MODO DE TESTBENCH

a) Estudiar el fichero progfile.dat que sirve para inicializar la memoria de programa de la cpu. Contiene un programa de ejemplo codificado usando la tabla anterior

```
0x0000 0100_0000_0000_0101          j Start
0x0001 0000_0000_0000_0000          nop
0x0002 0000_0000_0000_0000          nop
0x0003 0000_0000_0000_0000          nop
0x0004 0000_0000_0000_0000          nop
0x0005 0001_0000_0000_0010 Start:   li #0 R2       ;Registro destino del cálculo, 0->R2
0x0006 0001_0000_0010_0001          li #2 R1       ;Número de iteraciones, 2->R1
0x0007 0001_0000_0100_0011          li #4 R3       ;Valor a sumar a cálculo, 4->R3
0x0008 0001_0000_0001_0100          li #1 R4       ;Decremento unidad, 1->R4
0x0009 1010_0010_0011_0010 Iter:    add R2 R3 R2   ;suma, R2+R3->R2
0x000A 1011_0001_0100_0001          sub R1 R4 R1   ;resta uno del contador, R1-R4->R1
0x000B 0100_1000_0000_1001          jnz Iter
0x000C 0100_0000_0000_1100 Fin:     j Fin
```

Para observar su funcionamiento usaremos una implementación en Verilog del camino de datos y unidad de control ya compilada en el fichero cpu_tb.vpp. Realizamos una simulación de la cpu con ese programa en ejecución con el comando vvp:

```
vvp cpu_tb.vvp +n=14
```

donde la opción +n=14 especifica que la simulación durará 14 ciclos de reloj. Podemos observar las principales señales y los contenidos del banco de registros mediante el Gtkwave. Le pasamos el archivo de volcado de variables generado durante la simulación cpu_tb.vcd junto con un fichero de configuración cpu_tb.gtkw donde establecemos de antemano qué variables visualizar.

```
gtkwave cpu_tb.vcd cpu_tb.gtkw
```

Observar las señales de reset y reloj en relación con el valor del PC. Comprobar cómo evoluciona el PC según el flujo de control del programa del ejemplo. Chequear que las señales de control son producidas por la UC justo en el medio de cada ciclo de la instrucción correspondiente y que los cambios en los registros debidos a esas señales de control se producen al final del ciclo. Podemos codificar nuestro propio programa usando la tabla de más arriba y verificar su funcionamiento de la misma forma, posiblemente cambiando la opción +n=14 para simular un número mayor o menor de ciclos.

b) Estudiar y familiarizarse con el funcionamiento de los módulos suministrados: ALU, Banco de registros, multiplexores, registro PC, memoria de programa y realizar un nuevo módulo que represente el camino de datos representado más arriba, con la siguiente definición

```
module microc(output wire [5:0] Opcode, output wire z, input wire clk, reset, s_inc, s_inm, we3, wez, input wire [2:0] Op);
```

c) Completar un fichero testbench a partir de la estructura en microc_tb.v, que contenga código que permita realizar una simulación similar a la anterior. Partiendo del flujo de ejecución de las instrucciones del programa ejemplo (realizar una traza para ello), en el testbench iremos generando los valores de las señales de control de cada instrucción a ejecutar como si provinieran de la unidad de control (no es necesario crear un módulo específicamente para la unidad de control). Deberemos seguir la ejecución del programa hasta llegar a ejecutar un par de iteraciones del bucle infinito final (13 o 14 ciclos), emitiendo cada vez los valores de las señales de control correspondientes. Para hacer realista la simulación, supondremos que en la primera mitad del ciclo de reloj la unidad de control estaría ocupada decodificando la instrucción y que las señales de control se emitirían a partir de la mitad del ciclo hasta su fin, en el que recomienza el ciclo de la siguiente instrucción. Esto se conseguirá mediante la introducción de los retardos adecuados. Visualizar su correcto funcionamiento con el Gtkwave, los resultados deberían ser similares a los obtenidos en la simulación inicial.