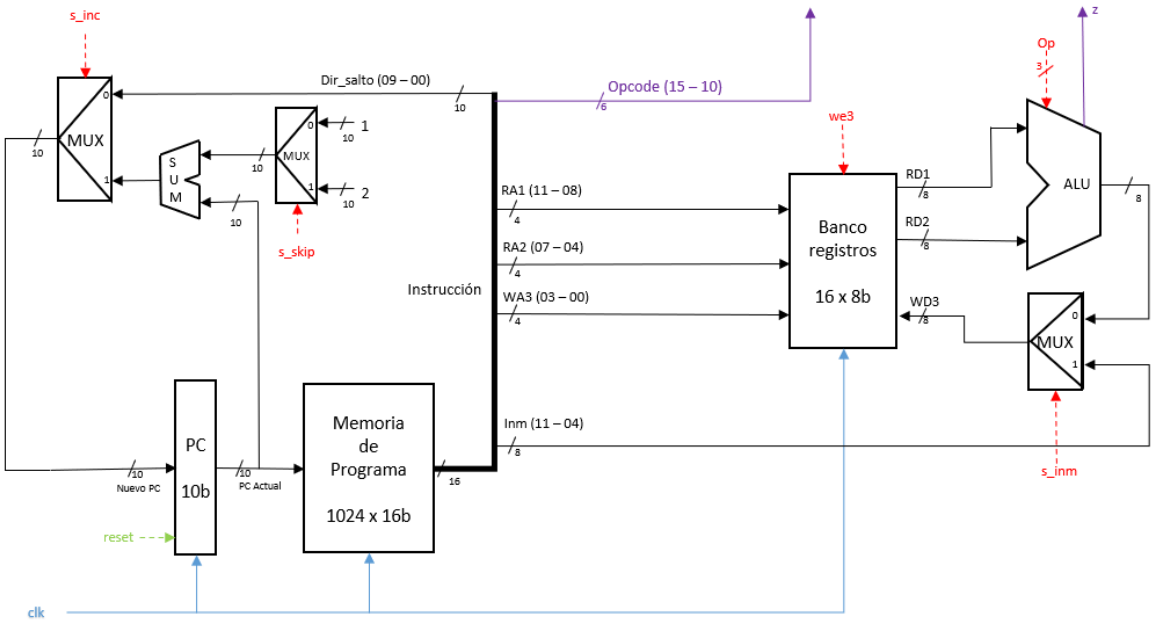
DESCRIPCIÓN DE LA MODIFICACIÓN

En esta práctica se continua el trabajo iniciado en la segunda sesión previa. El objetivo es plantear una modificación del camino de datos propuesto en esa actividad junto con los cambios correspondientes en el juego de instrucciones. Se realizará una simulación de una unidad de control generando las señales que emitiría durante la ejecución de un programa ejemplo propuesto. El nuevo camino de datos es el siguiente:



El camino de datos se diferencia del original en que podremos incrementar el PC en 1 o 2 unidades. La idea es que vamos a tener dos nuevas instrucciones **skipeq** y **skipne** que saltarán la siguiente instrucción (PC <- PC + 2) en función de la línea z que sale de la ALU (sustituyendo en el repertorio a las instrucciones jz y jnz). Para poderlas implementar hemos también eliminado el biestable FFZ conectado a la salida de la ALU, ya que en el mismo ciclo debe realizarse una resta y obtener el resultado del bit z de la ALU.

Instrucciones de salto condicional (SKIP): Opcode de 4 bits (15-12), campo de primer registro operando de 4 bits (11-8, RA1), campo de segundo registro operando de 4 bits (7-4, RA2). En los bits sobrantes pondremos "0001", lo que nos permitirá detectar errores en caso de que estemos realizando mal el control de estas instrucciones. En lugar de implementar un salto condicional al uso, nuestra CPU implementará instrucciones de "skip", que lo que hacen es saltarse la siguiente instrucción si se cumple la condición. Este tipo de instrucciones realizan una resta entre los dos operandos en binario puro, por lo que deberemos indicarle a la ALU que reste (ALUOp = 3'b011), aunque asegurándonos de deshabilitar la escritura en el banco de registros para evitar efectos colaterales (we3 = 0). Después comprueban el valor del flag z. Tenemos dos saltos diferentes:

SKIPEQ: Saltará si los dos registros son iguales, es decir si el flag de cero (zero) vale 1 tras la resta.

SKIPNE: Saltará si los dos registros no son iguales, es decir si el flag de cero (zero) vale 0 tras la resta.

En los casos en los que se cumpla la condición, debemos asegurarnos de que la señal s_inc esté a uno, para escoger la entrada del multiplexor que trae el incremento; y que la señal s_skip esté a uno, para sumarle dos unidades al PC, en lugar de una.

La siguiente tabla describe tanto la codificación (tal como aparecería en memoria de programa) como el funcionamiento de cada una de las instrucciones, incluyendo las dos nuevas. 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
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
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	0	1	0	0	0	D	D	D	D	D	D	D	D	D	D
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.																
Operación aritm./lógica (add, …)	1	Op	Op	Op	R1	R1	R1	R1	R2	R2	R2	R2	Rd	Rd	Rd	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.																
INSTRUCCIONES NUEVAS																
skip condicional si iguales (skipeq)	0	0	1	1	R1	R1	R1	R1	R2	R2	R2	R2	0	0	0	1
Opcode de 4 bits (15-12)), campo índice de primer registro operando de 4 bits (11-8, RA1), campo índice de segundo registro operando de 4 bits (7-4, RA2), 4 bits con "0001" (3-0). Se realizará la resta R1-R2 **(ALUOp = 3'b011)**, y en caso de ser (z=1 en la ALU), se incrementará el PC<-PC+2, en caso contrario, se incrementará PC<-PC+1. Debe tenerse en cuenta no escribir en el banco de registros.																
skip condicional si no iguales (skipne)	0	1	0	0	R1	R1	R1	R1	R2	R2	R2	R2	0	0	0	1
Opcode de 4 bits (15-12)), campo índice de primer registro operando de 4 bits (11-8, RA1), campo índice de segundo registro operando de 4 bits (7-4, RA2), 4 bits con "0001" (3-0). Se realizará la resta R1-R2 **(ALUOp = 3'b011)**, y en caso de ser (z=0 en la ALU), se incrementará el PC<-PC+2, en caso contrario, se incrementará PC<-PC+1. Debe tenerse en cuenta no escribir en el banco de registros.																

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

a) Estudiar y familiarizarse con el funcionamiento de los módulos suministrados: ALU, Banco de registros, multiplexores, etc. Realizar un nuevo módulo microc que represente el camino de datos dibujado más arriba, con la siguiente definición de interfaz:

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

b) Codificar el programa de abajo en el fichero progfile.dat, usando la tabla de codificación.

```
0x0000         jmp Start
0x0001         nop
0x0002         nop
0x0003         nop
0x0004         nop
0x0005 Start:  li  10, R2      ;Registro destino del cálculo
0x0006         li  4, R1       ;Número de iteraciones
0x0007         li  7, R3       ;Valor a sumar a destino de cálculo
0x0008         li  1, R4       ;Decremento unidad
0x0009 Iter:   add R2, R3, R2 ;R2 + R3 -> R2
0x000A         sub R1, R4, R1 ;R1 – R4 -> R1
0x000B         skipeq R1,R4
0x000C         jmp Iter
0x000D Fin:    jmp Fin
```

c) Completar un fichero testbench a partir de la estructura en el fichero microc_tb.v, que contenga código de la simulación de la emisión de las señales de control correspondientes a la ejecución del programa anterior. Partiendo del flujo de ejecución de las instrucciones del programa (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, 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, verificando que el PC evoluciona como estaba previsto en la traza.

ENTREGA

Al final de la sesión, después de haber mostrado los resultados a los profesores para su evaluación, se deberá entregar en la tarea del Campus Virtual creada para ello los ficheros microc.v, y microc_tb.v, así como cualquier fichero necesario para su funcionamiento y comprobación. Indicar en cada uno de los ficheros Verilog los nombres de los autores (máximo dos personas).