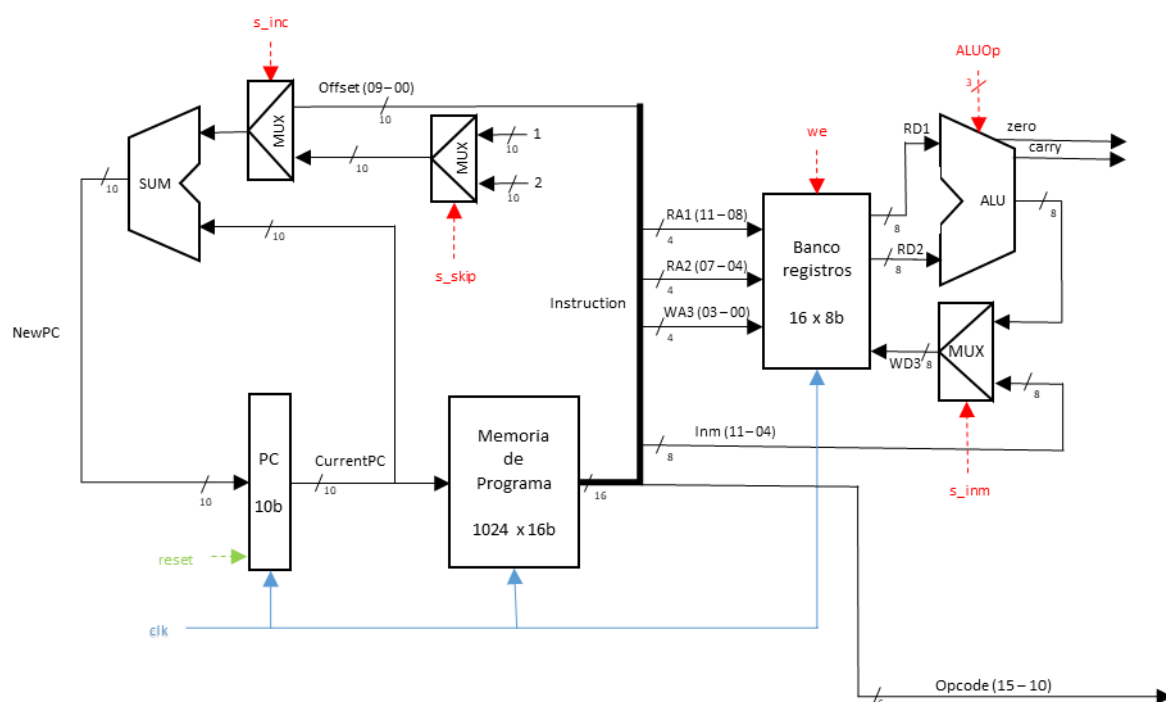


### DESCRIPCIÓN

El objetivo de la práctica es lograr una mejor comprensión sobre cómo funcionan el procesador y su unidad de control. Para ello, nos centraremos en un procesador muy simple de un sólo 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.



La figura representa el camino de datos del procesador. La parte superior izquierda muestra la lógica (sumador SUM y dos multiplexores) que se emplea para actualizar el contador de programa (PC). El PC, de 10 bits, sirve de dirección a la memoria de programa. El dato obtenido de esta memoria es la instrucción, de 16 bits. Esos 16 bits codifican varios tipos de instrucciones diferentes:

**Instrucción de carga de una constante inmediata:** *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 registro destino donde se escribirá la constante (WA3), siempre que el multiplexor que proporciona el dato a escribir tenga la entrada **s\_inm** a 1. El nuevo PC será el PC previo incrementado en 1, por lo que la señal **s\_inc** debe valer uno y la señal **s\_skip**, cero.

**Instrucción de operación aritmética o lógica:** *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) y campo de registro de destino de 4 bits (3-0) donde se almacenará el resultado (siempre que el multiplexor tenga **s\_inm** a cero). El nuevo PC será el PC previo incrementado en 1, como en el caso anterior.

**Instrucción de salto incondicional relativo:** *Opcod* de 6 bits (15-10) y los 10 bits restantes (9-0) serán el desplazamiento (*Offset*) mediante el cual se calcula el nuevo PC (nuevo PC = PC actual + desplazamiento). Para ejecutar un salto debemos hacer que el sumador use como entrada el desplazamiento; para ello, debemos poner un cero en la entrada de selección del MUX más a la izquierda (**s inc**).

**Instrucciones de salto condicional (SKIP):** *Opcod* 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 ( $we = 0$ ). Después comprueban el valor de los *flags*. Tenemos cuatro 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.
- SKIPGT: Saltará si el primer registro contiene un valor mayor que el segundo, es decir si el *flag* de acarreo (carry) vale 0 tras la resta.
- SKIPGE: Saltará si el primer registro contiene un valor mayor o igual que el segundo, es decir si el *flag* de acarreo (carry) vale 0 o el de cero (zero) vale 1 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.

Codificación	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
<b>Salto incondicional (JR)</b>	Op	Op	Op	Op	Op	Op	D	D	D	D	D	D	D	D	D	D
<b>Salto condicionales (SKIPEQ, SKIPNE, SKIPGT, SKIPGE)</b>	Op	Op	Op	Op	R1	R1	R1	R1	R2	R2	R2	R2	0	0	0	1
<b>Carga Inm. (LI)</b>	Op	Op	Op	Op	C	C	C	C	C	C	C	C	Rd	Rd	Rd	Rd
<b>Oper. ALU (ADD, SUB...)</b>	Op	Op	Op	Op	R1	R1	R1	R1	R2	R2	R2	R2	Rd	Rd	Rd	Rd

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 de los *flags* de cero y acarreo para la ejecución de los saltos condicionales

y generaría las salidas:

- señales de control de los multiplexores ( $s\_inc$ ,  $s\_skip$ , y  $s\_inm$ )
- la habilitación de escritura del banco de registros ( $we$ )
- señales de selección de operación de la ALU ( $ALUOp$ )

La misión de la unidad de control será activar correctamente las señales de salida a lo largo del ciclo que dura la instrucción de forma que se ejecute correctamente la instrucción determinada por los 6 bits (o menos) de *Opcode*.

## OBJETIVO: SIMULACIÓN DE LA UNIDAD DE CONTROL A MODO DE TESTBENCH

- Estudiar y familiarizarse con el funcionamiento de los módulos suministrados: ALU, Banco de registros, multiplexores, registro PC, memoria de programa. En particular, poner atención al fichero *progfile.dat* que sirve para inicializar la memoria de programa y el fichero *regfile.dat* que pone valores iniciales a los registros del banco de registros.
- Realizar un módulo que represente el camino de datos, con la siguiente definición

```
module microc(output wire [5:0] Opcode, output wire zero, carry, input wire clk, reset, s_skip, s_inc, s_inm, we, input wire [2:0] ALUOp);
```

- Realizar modificaciones al fichero *progfile.dat* y crear un fichero *testbench* *microc\_tb.v*, que contenga código que permita simular la ejecución de las instrucciones que el profesor indicará (de entre las anteriores). La idea es escribir un programa simple en binario en el fichero *progfile.dat* formado por las instrucciones pedidas. De acuerdo a dicho programa, en el *testbench* pedido generaremos las señales de control como si provinieran de una unidad de control (no es necesario crear un módulo específicamente para la unidad de control). 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 que recomienza el ciclo de la siguiente instrucción. Todo ello se conseguirá mediante la introducción de los retardos adecuados. Visualizar su correcto funcionamiento con el *Gtkwave*.

## 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 (dos personas).