# Trabajo de temas 3 (ALU y camino de datos) y 4 (Unidad de Control)

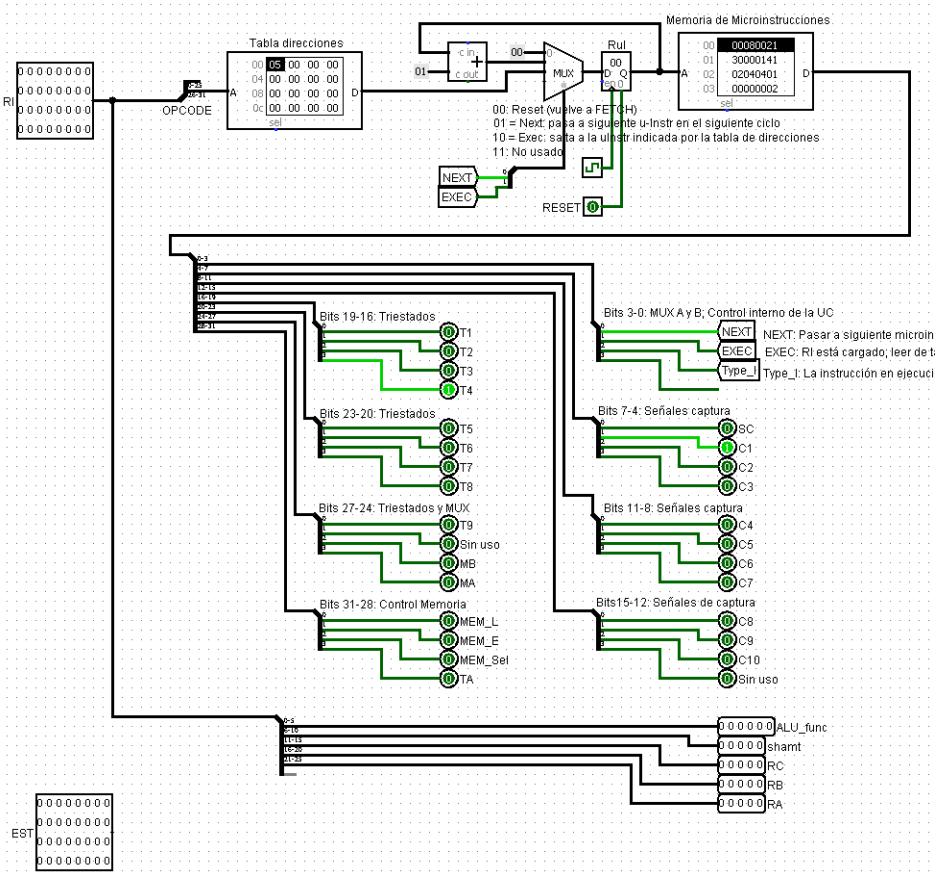
**Límite de entrega**: lunes 23 de mayo, 23:55 horas.

*Se penalizarán las entregas retrasadas con un cuarto de punto menos por cada hora de retraso.*

## Planteamiento del trabajo

Este trabajo se realiza sobre la versión más reciente disponible de la CPU MIPS con unidad de control, con la que ya hemos trabajado en prácticas.

Se permite realizar este trabajo en grupos de hasta tres componentes. Se hará una sola entrega en nombre de todo el grupo.



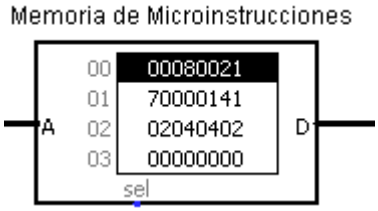

El objetivo principal del trabajo es aprender cómo funciona una UC **microprogramada**. Por eso la parte primera y principal del trabajo consiste en programar las microinstrucciones correspondientes a ciertas instrucciones de código ensamblador, y comprobar su funcionamiento.

Al igual que en el trabajo del tema 2 (programación en ensamblador MIPS), se ofrecen en este trabajo **diferentes niveles de completitud** con los que se puede aspirar a diferentes notas finales. La parte del trabajo que proporciona los 5 primeros puntos es muy sencilla porque solo hay que microprogramar instrucciones (y eso es casi idéntico a lo que ya habéis hecho en prácticas), pero a partir de ahí la dificultad aumenta porque se hace necesario diseñar o modificar circuitos.

## Primera parte: microprogramación de instrucciones

**Valor: 5 puntos.**

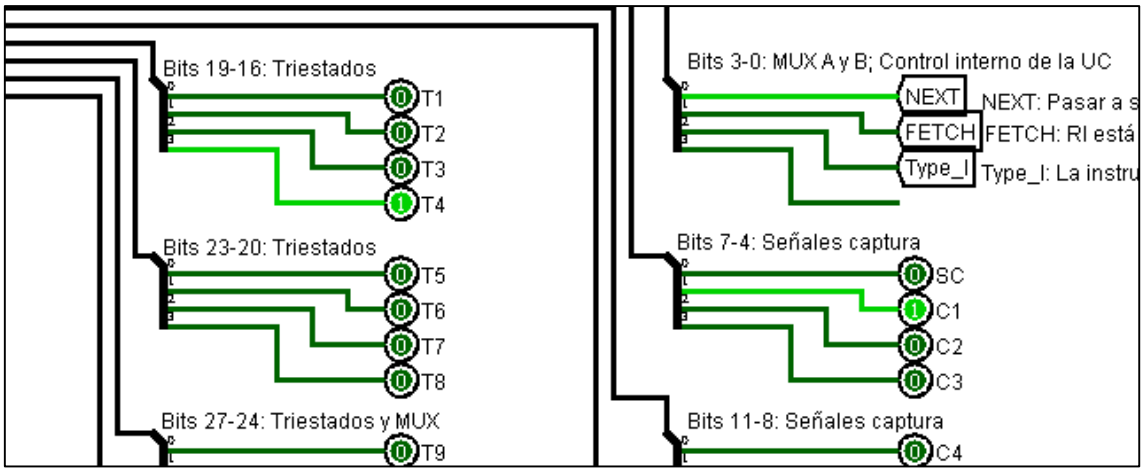
En la memoria de microinstrucciones se dan ya hechos los tres ciclos que componen la etapa de captación de instrucciones. Esto puede servir de ejemplo para ver cómo se expresa en forma hexadecimal una microinstrucción que se compone de 32 señales binarias.




Al avanzar mediante ciclos de reloj se verá cómo cambian las señales al seleccionarse diferentes microinstrucciones.

***NOTA**: En todo este trabajo no hay que preocuparse en ningún momento por la división entre primer y segundo semiciclo. Las señales y registros ya están configurados de manera que las capturas en registro capten los datos que se pongan a la entrada del registro en ese mismo ciclo.*

Para vuestra comodidad se han agrupado las señales de tal manera que cada grupo de 4 señales corresponde a un dígito hexadecimal de la microinstrucción:




Se pide en esta parte:

- Basándose en los ejercicios hechos en prácticas, determinar qué señales se tienen que activar en cada ciclo de ejecución de las instrucciones siguientes:
  - Aritmético-lógicas de tipo R (ej. ADD)
  - LW (tipo I)
  - SW (tipo I)
  - J (tipo J)
    - Importante: Las instrucciones de carga/almacenamiento tienen SIEMPRE un modo de direccionamiento constituido por un registro (Rt) y una constante (dato inmediato). La ejecución de estas instrucciones siempre comienza por ejecutar la suma de dicha constante con el contenido del registro indicado. El resultado se utilizará como puntero para indicar la posición de memoria que se quiere leer o escribir.

- Expresar en forma de microinstrucciones (en hexadecimal) cada una de esas instrucciones; una microinstrucción por cada ciclo de ejecución. Se puede usar el circuito generador de microinstrucciones que se utilizó en prácticas.
- Programar dichas microinstrucciones en la MµI de la misma manera que en las prácticas de unidad de control.
- Comprobar que cada una de las instrucciones se ejecuta de manera correcta. Para hacerlo, modificar manualmente el contenido inicial del RµI para que apunte a la microinstrucción que se quiere comprobar.
- Fuera de la UC, en la memoria principal del procesador, codificar en hexadecimal cuatro instrucciones en este orden:
    - ADD
    - LW
    - SW
    - J
        - Consejo: utilizando MARS se puede ensamblar un programa que tenga estas mismas instrucciones y comprobar que la codificación de las instrucciones es correcta.
- Escoger los operandos de cada una de manera que sea posible comprobar que de verdad están funcionando. Será necesario poner valores iniciales en algunos registros o posiciones de memoria.
  El salto incondicional de la última instrucción debe ser indicando el valor 0000001 como destino del salto (este valor se multiplica automáticamente por 4 al salir del RI a través de T9, lo que dará como destino del salto la dirección 000004, que es la de la segunda instrucción) para que entre así en bucle y ejecute {ADD, SW, J} indefinidamente.
- Comprobar que el bucle se ejecuta correctamente.
- Por precaución, guardar en fichero el estado de las memorias (sobre cada memoria, pulsar el botón derecho y escoger “Salvar imagen...”, guardar como .txt).
- Empleando un programa de captura de pantalla, grabar un vídeo **breve** en el que se aprecie el bucle en correcto funcionamiento. Se puede recibir hasta **1 punto extra** por narrar el vídeo dando una explicación clara del funcionamiento.
- Subir a la entrega los siguientes ficheros:
    - Fichero .circ en su estado final.
    - Ficheros .txt con el estado de cada memoria (memoria principal, tabla de direcciones, memoria de microinstrucciones).
    - Vídeo.

## Segunda parte: Operaciones con datos inmediatos

Valor: 1 punto

Se pide implementar las operaciones ADDI y SUBI (y opcionalmente otras artmético-lógicas de tipo I). Las instrucciones para hacer este añadido se han proporcionado en prácticas.

## Tercera parte: Multiplicación

Valor: 1 o 3 puntos

Se pide en este apartado implementar la instrucción MUL.

Para esto hay que añadir un multiplicador en la ALU. Este multiplicador puede ser combinacional (valor: 1 punto) o secuencial (valor: 3 puntos). En este apartado se permite expresamente

basarse en diseños encontrados en Internet (no obstante, no se permite importar directamente un multiplicador ya hecho en Logisim). En tal caso, se debe indicar la fuente.

Al igual que los demás apartados, para estar completo debe demostrarse su funcionamiento. Por tanto, deben hacerse las modificaciones necesarias en ALU, UC y memoria principal, de manera similar a la primera parte.

## Cuarta parte: Salto condicional

**Valor: 1, 2 o 3 puntos.**

Por un valor de un punto, se pide implementar la instrucción BEQ.
Esta instrucción es de tipo I. Se comparan mediante una resta los dos registros que refiere, y si sus contenidos son iguales, se salta a la dirección indicada por el dato inmediato. Esta instrucción emplea un tipo de direccionamiento que no hemos estudiado en clase; por ese motivo se permite simplificar su funcionamiento:

- El funcionamiento original emplea direccionamiento relativo al contador de programa (el valor que se capturará en el PC es el que resulta de multiplicar por 4 el dato inmediato y sumar este resultado al contenido actual del PC).
- El funcionamiento simplificado que se pide es que, en caso de ejecutar el salto, se guarde en PC el dato inmediato, sin operarlo antes.

Para implementar esta instrucción habrá que hacer una modificación de la UC. Son varias las maneras de implementar un salto condicional. Se dan como pistas estas ideas de diseño:

a) Implementar una función booleana que active o no active la señal C5 (captura en el PC, es decir, ejecutar salto) en función del opcode (¿salto incondicional, salto condicional? y de los bits de estado (¿resultado cero o no cero?).
b) Generar un reseteo del RμI si se detecta que la condición no se cumple (esto da por terminada la instrucción, y si se hace en un ciclo anterior a la captura en PC abortaría la ejecución del salto).
c) ¿...?

Al igual que en los otros apartados, para que se dé por bueno hay que demostrar su funcionamiento con la ejecución de una secuencia de instrucciones.

Puntos extra:

- 1 punto por implementar BNE además de BEQ.
- 1 punto por implementar SLT.

## Extra: Trabajo libre

Esta parte opcional es **individual**. En los ficheros que se entreguen se debe indicar claramente el nombre del autor o autora.

Se podrán recibir hasta 2 puntos por modificaciones y añadidos no recogidos en estos enunciados, por iniciativa de cada grupo. Ejemplos:

- Implementar otras instrucciones.
- Implementar otros operadores de la ALU.
- Demostrar la ejecución de un programa