1. ¿Cuáles son las tres partes de una CPU? ¿Cuáles de ellas forman lo que se conoce como camino de datos o *datapath*?

2. ¿Cuáles son las fases del ciclo de instrucción? Teniendo en cuenta la respuesta anterior y las instrucciones de MIPS indicar que elementos combinacionales y secuenciales podría necesitar una CPU para implementar dicha arquitectura.

3. Considerando el diagrama de la CPU de MIPS. ¿Para qué sirve el multiplexor antes de la entrada B de la ALU? Dar ejemplos de dos instrucciones donde se elija entre una u otra entrada del multiplexor.

4. Identificar qué unidades funcionales del *datapath* de MIPS intervienen en un `lw`, `sw`, `add`, `addi` y `beq`.

5. Sabiendo que en una computadora existen buses para datos, direcciones y control. Y que en el diagrama del *datapath* los cables del bus de control están resaltados en azul. Marcar en rojo los buses de direcciones y dejar en negro los buses de datos en el diagrama. ¿Encuentran alguna conexión que pueda interpretarse como de datos o direcciones según cambie la instrucción a ejecutar?

6. Considere la siguiente instrucción `and $t0, $t1, $t2`. ¿Qué valores toman las señales de control? ¿Qué unidades funcionales realizan una tarea útil? ¿Qué unidades producen una salida pero esa salida no se usa en la instrucción?

7. Si quisiéramos agregar la instrucción `sll` (*shif left logical*) al *datapath*. ¿Qué modificaciones tendríamos que hacer? ¿Qué unidades habría que agregar? En general para las operaciones de *shift* se usa un circuito llamado *barrel shifter*. Un ejemplo de la instrucción sería `sll $t0, $t1, 5` y significa desplazar a la izquierda 5 bits de `$t1` y guardar el resultado en `$t0`.

8. Asumiendo que los bloques del *datapath* tienen las siguientes latencias:

Instr Mem	Add	Mux	ALU	Reg File	Data Mem	Sign Ext	Shift Left 2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

   ¿Cuál es la duración mínima del ciclo de reloj de esta CPU? ¿Qué instrucción se ejecuta más rápido, un `add` o un `lw`? ¿Cuál es la frecuencia máxima de esta CPU y cuántas MIPS (millones de instrucciones por segundo) puede ejecutar?

9. Marcar el camino crítico para las instrucciones `lw`, `sw`, `add` y `beq` en el diagrama del *datapath* del libro de Patterson y Hennessy. El camino crítico es el camino más largo que recorre una instrucción de inicio a fin. Usar los tiempos de propagación del ejercicio anterior.

10. ¿Qué agregarían al *datapath* para implementar instrucciones de comparación como `slt` (*set on less than*)? Ejemplo: `slt $t0, $t1, $t2` significa si `$t1` es menor a `$t2` guardar un uno en `$t0`, de lo contrario guardar un cero.

11. Considere la instrucción 1010 1100 0110 0010 0000 0000 0001 0100. Asumiendo que la memoria de datos está completamente en ceros y que los registros tienen los siguientes valores.

$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
0	-1	2	-3	-4	10	6	8	2	-16

   (a) ¿Cuáles son la salidas del *sign extend* y del *shift left 2*?

   (b) ¿Qué valores toman las entradas de ALU control?

   (c) ¿Cuál es el nuevo valor del PC después de ejecutar la instrucción? Resaltar el camino por el cual se determina este valor.

   (d) Para cada multiplexor indicar el valor de su salida durante esta instrucción.

(e) ¿Qué valores toman las entradas de la ALU y de las dos unidades sumadoras?

(f) ¿Qué valores toman las entradas del archivo de registros?

12. Dibujar el camino que sigue una instrucción `beq` en el caso de que el *branch* sea verdadero. Dibujar solamente las unidades funcionales del *datapath* que intervienen.

13. ¿Qué modificaciones habría que hacer en el *datapath* para agregar la instrucción `jal` (*jump and link*)?

14. Mirando las señales de control para `beq`, `j`, `lw`, `sw` y las instrucciones de tipo R. ¿Existe la posibilidad de combinar dos o más señales? ¿Puede reemplazarse alguna señal por la inversa de otra?

15. Agregar a la unidad de control vista la instrucción `addi`. ¿Qué valores usarían para las dos señales de ALU Op?

16. ¿Qué diferencia encuentran en la palabra de control (las señales de control consideradas como un único número) entre estas dos instrucciones: `add $1, $2, $3` y `add $7, $8, $9`?

17. ¿Por qué es necesario tener una señal `RegWrite` para indicar que hay que escribir en el archivo de registros pero no hace falta una señal similar para el *Program Counter*?

18. Las 4 líneas que salen de ALU Control y entran como señales de control a la ALU, dentro de la ALU, ¿qué es lo que controlan?

19. Si solo quisiéramos implementar instrucciones de tipo R en la CPU de MIPS. ¿Qué señales de control eliminarían y por qué?

20. Imaginen que solo tenemos que implementar `sw`, `lw` y `addi` en la CPU de MIPS. ¿Cómo harían para implementar el control de este *datapath* sin agregar una sola compuerta para la unidad de control?

21. ¿Qué cambios harían en la unidad de control para que `jal` funcione correctamente?

22. Si eliminamos la señal `MemToReg`, ¿qué instrucciones no podrían ejecutarse correctamente?

23. La línea azul rotulada como `Zero` que sale de la ALU más que un dato es una señal de control que usa la instrucción `beq` para decidir si dos registros son iguales. Investigar y describir brevemente la solución que usa la arquitectura x86 que se remonta al microprocesador Intel 8080. Es decir, ¿cuál es la lógica en x86 que se usa para instrucciones como *branchs* que afectan a las instrucciones que siguen?

24. Realizar una tabla de doble entrada indicando en las columnas las cinco fases del *pipeline* de MIPS y en las filas las siguientes instrucciones: `lw`, `sw`, `beq`, `j`, `addi` e instrucciones de tipo R. Marcar con una cruz las etapas en las que cada instrucción hace algo útil.

25. Indicar el o los tipos de riesgos que aparecen en el siguiente código. Explicar en cada caso por qué se genera un riesgo.

```
  addi    $t1, $zero, 10
loop:
  beq     $zero, $t1, exit
  addi    $t1, $t1, -1
  addi    $v0, $zero, 1
  addi    $a0, $zero, 42
  syscall
  j       loop
exit:
  addi    $v0, $zero, 10
```

26. Considerar el siguiente código.

```
or $1, $2, $3
or $2, $1, $4
or $1, $1, $2
```

Indicar los riesgos de datos. Si no hay *forwarding* en este *pipeline*, ¿cómo podemos asegurar que las instrucciones den los resultados correctos? Dibujar un diagrama del *pipeline* para estas tres instrucciones teniendo en cuenta la solución propuesta para eliminar los riesgos.

27. Indicar los contenidos de los registros del *pipeline* y cuántos bits ocupa cada uno. Considerar las instrucciones `lw`, `sw`, `addi`, `beq` y las de tipo R.

28. ¿Por qué en un riesgo de datos entre dos instrucciones de tipo R no se necesita una "burbuja" pero entre un `lw` y una instrucción de tipo R sí?

29. Teniendo en cuenta el siguiente código:

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Encontrar los riesgos en el código y reordenar para evitar burbujas en el *pipeline*. Considerar qué riesgos pueden salvarse mediante *forwarding*.

30. Si insertamos burbujas al *pipeline* cada vez que en la fase IF traemos un *branch* para esperar a saber cuál es la próxima instrucción. ¿Cuántos ciclos de reloj desperdiciamos?

31. Siguiendo con la pregunta anterior. ¿Qué podemos hacer para minimizar el costo de parar el *pipeline* en cada *branch*?

32. Para los siguientes fragmentos de código indicar si: se ejecuta normalmente, se ejecuta correctamente usando solamente *forwarding* o hay que insertar burbujas incluso pudiendo hacer uso de *forwarding*.

(a)
```
lw  $t0, 0($t0)
add $t1, $t0, $t0
```

(b)
```
add  $t1, $t0, $t0
addi $t2, $t0, 5
addi $t4, $t1, 5
```

(c)
```
addi $t1, $t0, 1
addi $t2, $t0, 2
addi $t3, $t0, 2
addi $t3, $t0, 4
addi $t5, $t0, 5
```

33. Considere el siguiente código.

```
sub  $8, $9, $10
lw   $16, 8($8)
addi $4, $16, 8
add  $1, $3, $4
```

Dibujar el diagrama ilustrando como se ejecutan las instrucciones en el *pipeline* de MIPS teniendo en cuenta los riesgos y como se pueden solucionar.

34. Considere el siguiente código.

```
add  $4, $5, $6
addi $5, $6, 4
lw   $16, 100($5)
sub  $16, $16, $4
```

Dibujar el diagrama ilustrando como se ejecutan las instrucciones en el *pipeline* de MIPS teniendo en cuenta los riesgos y como se pueden solucionar.

35. ¿Por qué hay registros entre las etapas del *pipeline* de MIPS?

36. Simular en Logisim el *datapath* de MIPS usando *pipelining*. Ignorar completamente los riesgos de datos y de control. Agregar todas las señales de control necesarias pero no implementar la unidad de control.

37. Indicar verdadero o falso.
    (a) Una instrucción se ejecuta más rápido en un procesador con *pipeline* que sin él. ____
    (b) El archivo de registros puede ser utilizado por dos fases del *pipeline* de MIPS en el mismo ciclo de reloj. ____
    (c) Un procesador sin *pipelining* tiene un ciclo de reloj más largo que uno con *pipelining*. ____
    (d) En la versión de la CPU de MIPS con *pipelining* no hay multiplexores en la fase ID. ____
    (e) Un procesador que con un *pipeline* ejecuta más instrucciones por unidad de tiempo que uno sin él. ____
    (f) La CPU de MIPS con *pipeline* tiene las mismas señales de control que vimos en el *datapath* de ciclo único, incluso considerando los riesgos. ____

38. Completar los espacios en blanco.
    (a) El registro IF/ID guarda el _____ y la _____.
    (b) El *forwarding* ALU - ALU envía el _____ de la ALU del ciclo t a alguna de las _____ de la ALU del ciclo $t + 1$.
    (c) El cálculo de la dirección de la próxima instrucción requiere de al menos un _____. A lo sumo dos en el caso de un _____.
    (d) La señal de control _____ elige entre el resultado de una operación de la ALU o un dato de la _____ para escribir en los _____.
    (e) El registro MEM/WB guarda el dato de la _____, el resultado de _____, las señales de control _____ y _____ y el número de _____ a escribir.
    (f) La señal de control _____ elige entre una constante o un _____ como entrada B de la ALU.

39. En el *datapath* de MIPS. ¿Qué componente es el encargado de almacenar los resultados intermedios de las operaciones lógicas y aritméticas?
    ☐ La memoria de instrucciones.
    ☐ El archivo de registros.
    ☐ La ALU.
    ☐ La unidad de control.

40. ¿Cuáles son las líneas de control que valen 1 para un `lw`?
    ☐ RegDst, ALUSrc, MemToReg, MemRead
    ☐ ALUSrc, MemToReg, MemRead, ALU Op 1
    ☐ ALUSrc, MemToReg, MemRead, RegWrite
    ☐ RegDst, MemToReg, MemRead, RegWrite

41. ¿Cuáles de los siguientes elementos del *datapath* de MIPS interviene en un *jump*?
    - ☐ Contador de programa
    - ☐ ALU
    - ☐ Archivo de registros
    - ☐ Memoria de instrucciones

42. ¿Cuál de los siguientes no corresponde a un riesgo de un *pipeline*?
    - ☐ Estructural
    - ☐ De control
    - ☐ De datos
    - ☐ De direcciones

43. ¿Cuál de los siguientes elementos aparece al agregar *pipelining* al *datapath* de MIPS?
    - ☐ *Sign extend*
    - ☐ Mux de RegDst
    - ☐ *Hazard detection unit*
    - ☐ Mux de ALUSrc

44. La instrucción `beq` da lugar a un riesgo
    - ☐ Estructural
    - ☐ De control
    - ☐ De datos
    - ☐ De direcciones

45. La última fase del *pipeline* de MIPS visto es
    - ☐ MEM
    - ☐ WB
    - ☐ IF
    - ☐ ID

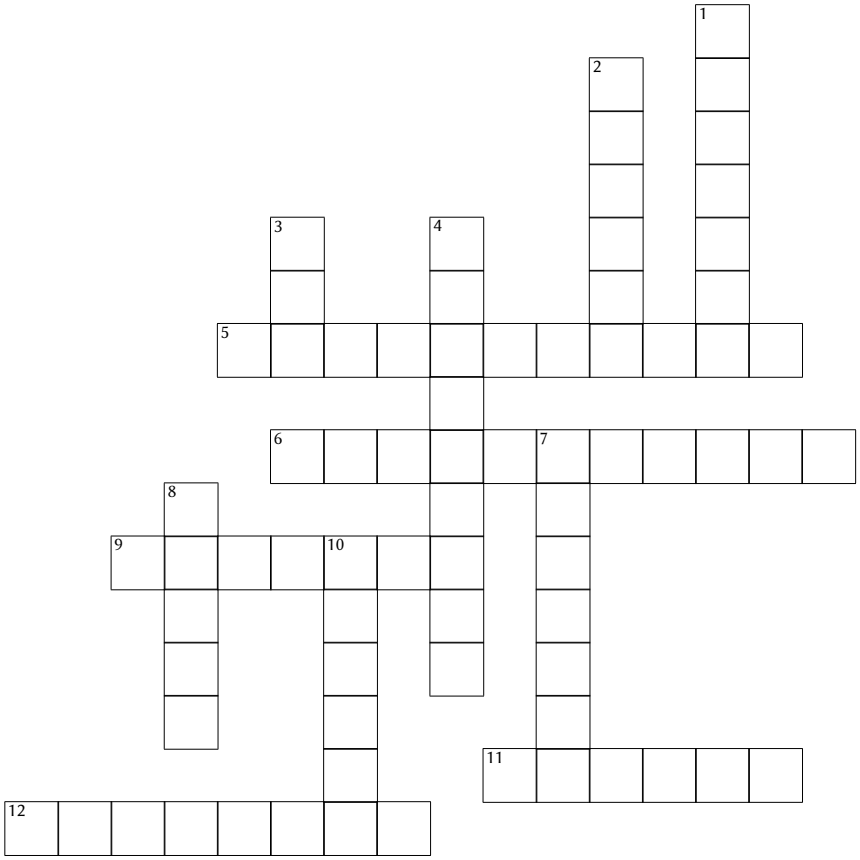
46. ¿Cuál de los siguientes elementos del *datapath* de MIPS no aparece al implementar *pipelining*?
    - ☐ *Forwarding unit*
    - ☐ Registro WB/IF
    - ☐ *Hazard detection unit*
    - ☐ Registro EX/MEM

47. ¿Cuál de los siguientes no corresponde a un tipo de bus?
    - ☐ Estructural
    - ☐ De control
    - ☐ De datos
    - ☐ De direcciones

48. La tercera fase del *pipeline* de MIPS visto es
    - ☐ MEM
    - ☐ WB
    - ☐ ID
    - ☐ EX

49. En el *pipeline* de MIPS. ¿Qué fase se encarga de detectar y resolver los riesgos de datos?
    - ☐ IF (*Instruction fetch*)
    - ☐ ID (*Instruction decode*)
    - ☐ EX (*Execute*)
    - ☐ WB (*Write back*)
    - ☐ HD (*Hazard detection*)

50. Completar el crucigrama.



Vertical 1 El camino más largo que sigue una instrucción 2 Entrada de la unidad de control 3 Se usa para ver si dos registros son iguales en un `beq` 4 La fase de la instrucción que modifica el archivo de registros 7 Las señales que eligen la operación a ejecutar 8 Todas las instrucciones son iguales en esta parte del ciclo de reloj 10 Vale uno en un `add` pero cero en un `addi`

Horizontal 5 Necesario cuando cuando hay más de una entrada posible para algún elemento del camino de datos 6 Lo que llevan los buses de la parte superior del diagrama, partiendo del PC 9 Solo acceden a ella las instrucciones `lw`, `sw` o similares 11 Vale cero cuando realizamos una cuenta entre dos registros 12 El conjunto de Unidad Aritmético Lógica y registros