1. Escribir un programa en MIPS que imprima en consola “Hola MIPS”.
2. Escribir un programa en MIPS que imprima en consola “Hola, `<arg>`”, dónde `<arg>` es un argumento de línea de comandos.
3. Escribir código en ensamblador de MIPS equivalente a los siguientes enunciados de C.
   (a) `int a = 12;`
   (b) `b = b - 1;`
   (c) `c++;`
   (d) `d /= 10;`
   (e) `e = e % 2;`
   (f) `f = g + (h - 5);`
   (g) `i *= 10;`
   (h) `j = i ^ 1600;`
   (i) `k = i & (j | k);`
4. Dar las instrucciones en C equivalentes al siguiente código de ensamblador de MIPS. Asumir que las variables `f`, `g`, `h`, `i` y `j` están asignadas a los registros `$s0` a `$s4`. Las direcciones base de los *arrays* `A` y `B` están en los registros `$s6` y `$s7`.

```
sll  $t0, $s0, 2    # $t0 = f * 4
add  $t0, $s6, $t0  # $t0 = &A[f]
sll  $t1, $s1, 2    # $t1 = g * 4
add  $t1, $s7, $t1  # $t1 = &B[g]
lw   $s0, 0($t0)    # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

5. ¿Qué instrucción de MIPS sirve para sumar un registro y una constante?
   - ☐ `sum`
   - ☐ `add`
   - ☐ `addi`
   - ☐ `addc`
6. ¿Cuál de los siguientes no es un tipo de instrucción en MIPS?
   - ☐ R
   - ☐ S
   - ☐ J
   - ☐ I
7. ¿Cuál de las siguientes no es una instrucción real en MIPS?
   - ☐ `li`
   - ☐ `la`
   - ☐ `move`
   - ☐ Todas son pseudoinstrucciones.

8. En ensamblador de MIPS las palabras que comienzan con un punto son
   - ☐ Etiquetas
   - ☐ Directivas
   - ☐ Registros
   - ☐ Instrucciones

9. ¿Cuál de las siguientes instrucciones sirve para multiplicar un número por cuatro?
   - ☐ `sll $s0, $s0, 2`
   - ☐ `sll $s0, $s0, 4`
   - ☐ `srl $s0, $s0, 2`
   - ☐ `srl $s0, $s0, 4`

10. ¿Cuál de las siguientes instrucciones en C es equivalente a `and $t0, $t0, $zero`?
    - ☐ `a && 0`
    - ☐ `a = a && 0`
    - ☐ `a &= 0`
    - ☐ `a & b`

11. ¿Para qué sirven las etiquetas en ensamblador?
    - ☐ Para indicar dónde deben ubicarse en memoria los contenidos del archivo de código fuente.
    - ☐ Para no referirse a direcciones de memoria por números sino por símbolos.
    - ☐ Para separar las instrucciones de los datos.
    - ☐ Para declarar variables.

12. ¿Cuál de las siguientes directivas indica que lo que sigue es un número entero de 32 bits?
    - ☐ `.word`
    - ☐ `.asciiz`
    - ☐ `int:`
    - ☐ `.data`

13. ¿Qué registro en MIPS apunta al tope de la pila (*stack*)?
    - ☐ `$gp`
    - ☐ `$fp`
    - ☐ `$sp`
    - ☐ `$ra`

14. ¿Qué registro usarías para devolver el valor de una función en MIPS?
    - ☐ `$s0`
    - ☐ `$v0`
    - ☐ `$a0`
    - ☐ Cualquier registro sirve.

15. ¿Cuál es la función de los registros `$s0` a `$s9`?
    - ☐ Son registros para pasar argumentos a una función.
    - ☐ Son registros de propósito general.
    - ☐ Son registros utilizados por el sistema operativo.
    - ☐ Son registros utilizados para realizar cuentas.

16. ¿En qué dirección de memoria se ubica el código del programa según la segmentación de memoria usada en MIPS?
    - ☐ `0×1000 0000`
    - ☐ `0×0040 0000`
    - ☐ `0×1000 8000`
    - ☐ `0×0000 0000`

17. ¿Qué código numérico se usa para indicar al sistema operativo que queremos imprimir un número entero?
    - ☐ 1
    - ☐ 2
    - ☐ 4
    - ☐ 10

18. ¿Dónde debemos poner la dirección del *string* que queremos imprimir al usar una *system call*?
    - ☐ `$s0`
    - ☐ `$v0`
    - ☐ `$a0`
    - ☐ `$k0`

19. En una instrucción *load word* que función cumple el valor del campo `imm`?
    - ☐ Es el número que queremos cargar en el registro.
    - ☐ Es un *offset* de la dirección base.
    - ☐ Es un código que indica el registro dónde queremos cargar el valor.
    - ☐ Es la dirección de memoria desde la cual queremos cargar el valor.

20. Los campos de una instrucción de tipo R son
    - ☐ `op rs rt rd shamt funct`
    - ☐ `op rd rs rt shamt funct`
    - ☐ `op rd rs rt imm funct`
    - ☐ `op rs rt imm`

21. El rango de números válidos para el campo *immediate* es de
    - ☐ $[-2^{15}, 2^{15}]$
    - ☐ $[-2^{31}, 2^{31} - 1]$
    - ☐ $[-2^{16}, 2^{16} - 1]$
    - ☐ $[-32768, 32767]$

22. ¿Qué instrucción usa el ensamblador de MIPS en lugar de `li $v0, 4`?

    - ☐ `ori $2, $0, 4`
    - ☐ `ori $zero, $v0, 4`
    - ☐ `add $0, $2, 4`
    - ☐ `lw $v0, 4`

23. ¿Cuál de los siguientes registros no deben ser preservados en el prólogo de una función?

    - ☐ `$sp`
    - ☐ `$ra`
    - ☐ `$s0`
    - ☐ `$t0`

24. ¿En qué caso una función no tiene la necesidad de preservar el valor del *return address*?

    - ☐ En ningún caso.
    - ☐ Cuándo lo hace el *caller*.
    - ☐ Cuándo es una *leaf procedure*.
    - ☐ Cuándo es una función no recursiva.

25. Escribir un programa en MIPS que acepte dos números enteros por entrada estándar, un dividendo y un divisor. Realizar la división entera de los dos números.

    Mostrar como salida del programa la siguiente ecuación: dividendo = divisor × cociente + resto.

26. Escribir un programa en MIPS que acepte como entrada un número entero de días. El programa debe imprimir la cantidad de años, semanas y días correspondiente. Ignorar los años bisiestos. Por ejemplo para 375 días como entrada, el programa imprime: "1 año, 1 semana, 3 días".

27. Escribir un programa en MIPS que acepte tres argumentos por línea de comandos e imprima un saludo para los tres nombres ingresados al revés de como fueron ingresados. Por ejemplo:

```
$ spim -q -f hola.s Juan Pedro Maria
Hola Maria, Pedro y Juan
```

28. Escribir un programa en MIPS que intercambie el valor de dos variables enteras *a* y *b*. Los valores iniciales de *a* y *b* se obtienen por entrada estándar. Imprimir los valores de *a* y *b* antes y después del cambio.

29. Escribir un programa en MIPS que lea un número desde memoria a un registro e invierta todos sus bits como si se tratara de una compuerta NOT.

    Cabe destacar que no existe la instrucción `not` en el conjunto de instrucciones de MIPS.

30. Escribir un programa en MIPS que muestre el valor absoluto de un número ingresado por entrada estándar.

31. Escribir un programa en MIPS que decida si un número es par o no.

32. Escribir un programa en MIPS que acepte tres argumentos enteros e imprima "iguales" si los tres números son iguales o "no iguales" de lo contrario. Dar el diagrama de flujo.

33. Escribir un programa en MIPS que decida si un año es bisiesto o no.

34. Escribir un programa que reciba dos números enteros positivos e imprima "verdadero" si alguno de los argumentos es divisible por el otro.

35. Escribir un programa en MIPS que acepte 5 números enteros e imprima la suma solamente de los argumentos que sean impares.

36. Escribir un programa en MIPS que acepte dos números como entrada: día y mes.

    Imprimir "verdadero" si la fecha está entre el 20 de marzo y el 20 de junio, o "falso" de lo contrario.

37. Escribir un programa en MIPS que acepte dos números enteros x, y representando coordenadas en el plano cartesiano. El programa debe imprimir a qué cuadrante pertenece el punto. Los cuadrantes del plano cartesiano se representan con los números romanos I, II, III y IV.

38. Escribir un programa en MIPS que sume los números del 1 al 100.

39. Escribir un programa en MIPS que imprima los números entre el 1000 y el 1999, mostrando 5 números por línea.

40. Escribir un programa que haga lo mismo que la función `atoi()` de la librería estándar de C.

    Es decir que reciba como entrada un *string* de caracteres numéricos y lo convierta a su representación como número entero.

41. Escribir un programa en MIPS que acepte números ingresados por el usuario hasta que el usuario ingrese el número cero. La salida del programa es la suma de todos los números ingresados.

42. Escribir un programa en MIPS que reciba un número n por entrada estándar e imprima n veces la frase "I am Groot". Si $n \leq 0$ imprimir "Error".

43. Escribir un programa en MIPS que acepte un número entero n como argumento y calcule $n!$, el factorial de n. Si $n < 0$ imprimir "Error".

44. Escribir un programa en MIPS que acepte dos argumentos enteros a y b con $b > a$. El programa debe imprimr todos los números divisibles por 3 en el intervalo $[a, b]$. Si a es mayor a b el programa debe imprimir un mensaje indicando al usuario el uso correcto del programa y salir.

45. Escribir un programa que acepte un argumento entero n e imprima los números de 1 hasta n y su cuadrado. Por ejemplo:

```
$ spim -q -f squares.s 3
1 → 1
2 → 4
3 → 9
$ spim -q -f squares.s 5
1 → 1
2 → 4
3 → 9
4 → 16
5 → 25
```

46. Escribir un programa en MIPS que acepte como argumento un número entero e imprima el número con los digitos al revés. Por ejemplo para la entrada 12345 debe imprimir 54321.

47. Escribir un programa en MIPS que acepte como argumento un *string* y cuente la cantidad de caracteres que tiene.

48. Escribir un programa en MIPS que reciba como argumento un *string* y cambie las mayúsculas por minísculas y viceversa.

49. Escribir un programa en MIPS que acepte un número n por entrada estándar e imprima 2^n por consola. Si el número es menor a cero imprimir "Error" y salir del programa.

50. Escribir un programa en MIPS que decida si un entero positivo n es primo o no. Un número es primo si es divisible sólo por uno y por sí mismo. Por ejemplo 7 es primo pero 33 no.

51. Usando el algoritmo de Euclides para encontrar el MCD (máximo común divisor) escribir un programa en MIPS que acepte como argumentos dos números enteros a y b e imprima su MCD. El algoritmo de Euclides en pseudocódigo sería así:

    while $b \neq 0$ **do**
    $t \leftarrow b$
    $b \leftarrow a \mod b$
    $a \leftarrow t$
    end
    print a

    El valor final de a es el resultado del algoritmo, es decir el MCD de a y b.

52. Escribir un programa en MIPS que use un ciclo para imprimir los primeros n números de Fibonacci, siendo n un argumento entero del programa. La sucesión de Fibonacci está definida de la siguiente manera.

    $$F_0 = 0,\ F_1 = 1$$

    Es decir los dos primeros números son cero y uno. A partir del tercer número (F_2) y en adelante cada número es la suma de los dos anteriores.

    $$F_n = F_{n-2} + F_{n-1}$$

    Por ejemplo, los primeros diez números de Fibonacci $F_0 \dots F_9$ son:

    $$0, 1, 1, 2, 3, 5, 8, 13, 21, 34$$

53. Escribir un programa en MIPS que acepte un entero positivo n como argumento e imprima el mismo número en binario. Para realizar la conversión tener en cuenta que un número en binario está expresado como suma de potencias de dos. Primero buscar la potencia de dos más grande que entra en el número a convertir. Ese es el uno que está más a la izquierda. Luego ir probando con las potencias de dos sucesivas en orden decreciente. Si sumamos una potencia y no nos pasamos del número entonces va un uno, si nos pasamos va un cero. Repitiendo estos pasos obtenemos la representación de n en binario.

54. Escribir un programa en MIPS que realice la factorización de enteros en factores primos. Ejemplos:

    ```
    $ ./factores 60
    2 2 3 5
    $ ./factores 72
    2 2 2 3 3
    ```

    Por el teorema fundamental de la aritmética todo número natural tiene una descomposición única en factores primos. Por ejemplo:

    $$72 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3$$

55. Escribir un programa en MIPS que imprime la matriz identidad de $n \times n$. Usar dos *loops* anidados con variables de control i y j.

56. Escribir un programa en MIPS que decida si un número entero es positivo, negativo o cero.

57. Escribir un programa en MIPS que decida si un *string* ingresado como argumento es un palíndromo o no.

    Un palíndromo es una palabra que se lee igual de derecha a izquierda que de izquierda a derecha como “neuquen” o “reconocer”.

58. Escribir un programa en MIPS que reciba como argumento una palabra y una letra y cuente la cantidad de veces que aparece esa letra en la palabra.

59. Escribir un programa en MIPS que acepte un argumento entero n y use dos *loops* anidados para imprimir un patrón como el de un tablero de ajedrez usando asteriscos y espacios en la terminal. El argumento n es la cantidad de filas y columnas del tablero.

```
$ spim -q -f chess.s 3
* *
 *
* *
$ spim -q -f chess.s 8
* * * *
 * * * *
* * * *
 * * * *
* * * *
 * * * *
* * * *
 * * * *
```

60. Convertir el ejercicio 40 en una función. Es decir, dar una versión en ensamblador de MIPS de `int atoi(char *str)`.

61. Dar una versión en ensamblador de MIPS de `int length(char *str)`, que devuelve la longitud de un *string*.

62. Dar una versión en ensamblador de MIPS de `int isDigit(char l)`, que devuelve 1 si un caracter es dígito o 0 de lo contrario.

63. Dar una versión en ensamblador de MIPS de `int isAlpha(char l)`, que devuelve 1 si un caracter es una letra o 0 de lo contrario.

64. Dar una versión en ensamblador de MIPS de `void toLowerCase(char *str)`, que pasa un *string* a minísculas, dejando todo caracter que no sea letra o que ya esté en minísculas sin alterar.

65. Dar una versión en ensamblador de MIPS de `void toUpperCase(char *str)`, que pasa un *string* a mayúsculas, dejando todo caracter que no sea letra o que ya esté en mayúsculas sin alterar.

66. Escribir una función `int isPalindrome(char *str)` que devuelva 1 si una palabra es palíndromo o 0 de lo contrario. Ignorar mayúsculas y minísculas.

67. Escribir las siguientes funciones en ensamblador de MIPS.
    (a) Una función que devuelva 1 si dos números a y b son iguales o 0 de lo contrario.
    (b) Una función que devuelva el valor absoluto de un número a.
    (c) Una función que devuelva el máximo de dos números a y b.
    (d) Una función que devuelva el mínimo de dos números a y b.
    (e) Una función que devuelva el promedio de dos números a y b.
    (f) Una función que devuelva el producto escalar entre dos vectores en el plano.
    (g) Una función que reciba tres longitudes a, b y c, decidir si se puede formar un triángulo con esos números.

68. Escribir las siguientes funciones de manera iterativa y recursiva. En todos los casos $n \in \mathbb{N}$
    (a) $f(n) = \sum_{i=1}^{n} n$
    (b) $f(b, n) = b^n$
    (c) $f(n) = n!$
    (d) $f(n) = f(n-1) + f(n-2)$ con $f(0) = 0$ y $f(1) = 1$

69. Traducir las siguientes instrucciones de MIPS en ensamblador a código máquina (en hexadecimal).
    (a) `add $t0, $t0, $zero`
    (b) `add $t0, $s2, $t0`
    (c) `lw $t1, 4($s3)`
    (d) `lw $t0, 1200($t1)`
    (e) `lb $t0, 1200($t1)`
    (f) `sw $t0, 1200($t1)`
    (g) `sll $s0, $s0, 4`
    (h) `lw $s0, 4($a1)`
    (i) `or $a0, $a1, $zero`
    (j) `sb $t0, 1200($t1)`
    (k) `sw $t1, 32($t2)`
    (l) `li $v0, 10`
    (m) `syscall`
    (n) `addi $v0, $zero, 10`
    (o) `xori $s0, $s1, 32`
    (p) `sub $a0, $t2, $t1`
    (q) `addi $sp, $sp, -32`

70. Pasar las siguientes instrucciones de MIPS en binario a lenguaje ensamblador.
    (a) `1010 1110 0000 1011 0000 0000 0000 0100`
    (b) `1000 1101 0000 1000 0000 0000 0100 0000`
    (c) `0000 0010 0001 0000 1000 0000 0010 0000`
    (d) `1010 1100 0110 0010 0000 0000 0001 0100`
    (e) `0010 0010 0010 0010 0000 0000 0011 1100`

71. Pasar las siguientes instrucciones de MIPS en hexadecimal a lenguaje ensamblador.
    (a) `0×8D08FFC0`
    (b) `0×8D090012`
    (c) `0×AE0BFFFC`
    (d) `0×01095022`
    (e) `0×0000000C`
    (f) `0×08400000`
    (g) `0×38440020`
    (h) `0×00901025`

72. Determinar a qué instrucción en lenguaje ensamblador corresponden los siguientes valores de los distintos campos de una instrucción.
    (a) `op = 0, rs = 1, rt = 2, rd = 3, shamt = 0, funct = 32`
    (b) `op = 0, rs = 3, rt = 2, rd = 3, shamt = 0, funct = 34`
    (c) `op = 0×2B, rs = 0×10, rt = 0×5, imm = 0×4`
    (d) `op = 0×23, rs = 1, rt = 2, imm = 0×4`
    (e) `op = 0, rs = 0, rt = 9, rd = 8, shamt = 5, funct = 0`

73. Si quisiéramos expandir la cantidad de registros a 128 y tener 4 veces la cantidad de instrucciones en MIPS.
    (a) ¿Cómo afectaría este cambio el tamaño de los campos de las instrucciones tipo R?
    (b) ¿Cómo afectaría este cambio el tamaño de los campos de las instrucciones tipo I?

74. Suponiendo que el PC vale `0×2000 0000`.
    ¿Es posible usar la instrucción `j` (*jump*) para setear el PC en `0×4000 0000`? ¿Es posible hacerlo con un `beq`?

75. La siguiente instrucción no existe en MIPS.
    `rpt $t2, loop # if (R[rs]>0) R[rs]=R[rs]-1, PC=PC+4+BranchAddr`
    (a) ¿Cuál es el formato más apropiado para esta instrucción?
    (b) Dar una secuencia mínima de instrucciones en ensamblador que realicen la misma operación.

76. Traducir a código máquina el siguiente fragmento de código, indicando las direcciones de memoria de cada instrucción.

```
.text
main:
  li $s0, 10
  li $v0, 1
loop:
  move $a0, $s0
  syscall
  addi $s0, $s0, -1
  beq $s0, $zero, exit
  j loop
exit:
  li $v0, 10
  syscall
```

77. Traducir a código máquina el siguiente fragmento de código, indicando las direcciones de memoria de cada instrucción.

```
.text
main:
  lw $s0, 4($a1)
loop:
  lb $s1, 0($s0)
  beq $s1, $zero, exit
  xori $s1, $s1, 32
  sb $s1, 0($s0)
  addi $s0, $s0, 1
  j loop
exit:
  li $v0, 10
  syscall
```