A) Viendo que en assembler no puede sumar directamente 4228 por ser un número muy grande, utiliza 2 líneas para hacer la suma total, o sea 2 posiciones de memoria, teniendo en cuenta que esto también sucede con la asignación “li a1, 2114” las etiquetas tendrán las siguientes posiciones de memoria:
-fin: posicion de memoria 14
-resta: posición de memoria 18
-sigo: posición de memoria 28
-epílogo: posición de memoria 2c

B) inicio -> resta: Prologo -> sigo -> prologo -> epilogo -> epilogo -> fin
C) El rango de uso de li es para numeros de hasta 32 bits a diferencia del addi que solo permite 12.
D) lo resuelve usando lui que se encarga de copiar los valores desde el bit 13 al 32 y después addi (que tiene capacidad de 12 bits) suma los 12 bits que no copio lui.
E) valor final de a1: 2114 o 0x0000842 o 100001000010
F) valor final de PC: 0x14
G) Secuencia del PC: 0x0 -> 0x04 -> 0x08 -> 0x0c -> 0x10 -> 0x18 -> 0x1c -> 0x20 -> 0x24 -> 0x28 -> 0x18 -> 0x1c -> 0x20 -> 0x24 -> 0x2c -> 0x30 -> 0x34 -> 0x2c -> 0x30 -> 0x34 -> 0x14
H) Ra: 0x0 -> 0x14 -> 0x2c -> 0x14
SP: 0x7ffffff0 -> 0x7fffffec -> 0x7fffffe8 -> 0x7fffffec -> 0x7ffffff0

I)
```
li a0,4228
srai a1, a0 ,1
jal ra, resta
fin: beq zero, zero, fin
resta:
  prologo:addi sp,sp ,-4 #guarda palabra en el sp
        sw ra, 0(sp) # guarda el ra en la pila
        sub a0, a0, a1
        beq a0, zero, epilogo
  sigo:   jal ra, resta
  epilogo:
        lw ra, 0(sp) # ultimo valor de la pila?
        addi sp, sp, 4 # desplaza a la palabra de la lista
        ret #voy al proceso en el cual apunta ra
```

Codigo donde se reemplazo a1 por la mitad de a0 en una unica instruccion, con “srai a1, a0, 1” guardamos en a1 la mitad de a0 corriendo el bit mas significativo de izquierda a derecha.

```
main:
   addi x11 x0 4
   lw x12 0 x11
   addi x13 x0 4
   lw x13 0 x13
   lw x13 0 x13
   beq x12 x13 -20 #como x12 y x13 son distintos entonces no hace el -20
   #el -20 simboliza ir 5 instrucciones para atras ya que -20 = 4x-5

guardar:
    lui x14 0xfffa6
    addi x14 x14 -1539
    add x12 x14 x12
    sw x11 40 x12
fin:
   addi x10 x0 0
   addi x17 x0 93
   ecall
```

3)A)
1: modifica a1 para que valga 4
2: le asigna a a2 el valor en la posicion de memoria a1(que es 4), por consigna asumimos que este valor es 0
3: idem con item 1 reemplazando por a3
4: idem con item 2 reemplazando por a3
5: le asigna a a2 el valor de la posicion de memoria a3 (que por el item anterior sabemos que a3 = 0), por consigna asumimos nuevamente que el valor es 0
6: hace la comparación de a2 con a3 y dado que ambos son 0 pisa el pc y lo lleva devuelta a la primera instrucción del programa (con valor de pc = 8). Creando un bucle sin salida

B) La modificación es que ahora sabemos que en la posición de memoria 4 hay una instrucción, por lo que en los items 2 y 4 ahora quedarían a2 y a3 iguales al código hexa de esta instrucción. Pero, en el item 5 cambia ya que ahora busca asignarle a a3 lo que hay en la posición de memoria correspondiente a a3 (que sabemos que ahora tiene una instrucción asignada) por lo que al no tener nada asignado en esta nueva posición de memoria que llamamos a3 pasa a ser igual a 0. Esto genera que a2 y a3 sean distintos cuando llega al item 6, evitando el bucle.
En los siguientes pasos termina de hacer las dos etiquetas que quedan:

- guardar: se encarga de guardar en la posicion de memoria 40(a2) (ahora a2 vale 0 ya que se le sumo el inverso negativo)
- fin_programa: con ecall se encarga de terminar el programa y devolver el codigo de error 0 por consola

Cosas revisar: Pasar ejercicio 2 a la planilla.
Ejercicio 3b: porque retorna 0 cuando termina el programa
ejercicio 3b: preguntar la logica de sw