(1) Introducción - Recorrer la máquina y la hoja de datos, y responder:

a) ¿Qué componentes están involucrados en cada paso del ciclo Fetch - Decode - Execute de la máquina OrgaSmallI?

En este diseño, tanto la operación de fetch como decode son ejecutadas por medio de micro-instrucciones. Una vez identificada la instrucción a ejecutar, la operación de execute también es realizada mediante un código de micro-instrucciones. Inicialmente microPC vale cero, posición en la que se almacena la secuencia de pasos correspondientes a la etapa de fetch. En esta se carga desde la memoria la próxima instrucción a ser ejecutada indicada por el **PC**. Como la **memoria** direcciona a byte, y las instrucciones ocupan dos bytes, el proceso de fetch debe cargar dos valores desde memoria. Estos valores son enviados a la **unidad de decodificación** (DE). Esta última genera cuatro salidas M, X, Y y OP. Las primeras tres corresponden a los parámetros de la instrucción y la restante a su opcode. Luego la **UC** carga en el microPC el valor opcode << 4, es decir, agrega cuatro ceros en los bits menos significativos del opcode. Las siguientes micro-instrucciones corresponden a la secuencia de señales para resolver la instrucción indicada por el opcode. Una vez resuelta la instrucción, el ciclo comienza nuevamente seteando microPC en cero.
Si el dispositivo de E/S activa la **señal de interrupción** y el flag I vale 1, termina de ejecutar la instrucción en curso y realiza atómicamente los siguientes pasos:
Coloca [0xFF] = PC
Coloca I=0 para evitar que el procesador vuelva a interrumpirse
Coloca PC = [0x00]
Activa la señal INTA para indicarle al dispositivo que atenderá su pedido
Luego, comienza a ejecutarse la rutina de atención de la interrupción propiamente dicha.

b) ¿En qué parte del ciclo de instrucción (Fetch - Decode - Execute) deberían detectarse y atenderse las interrupciones?

En el caso de un procesador OrgaSmallI, si el dispositivo de E/S activa la señal de interrupción y el flag I vale 1, termina de ejecutar la instrucción en curso y realiza atómicamente los siguientes pasos:
Coloca [0xFF] = PC
Coloca I=0 para evitar que el procesador vuelva a interrumpirse
Coloca PC = [0x00]
Activa la señal INTA para indicarle al dispositivo que atenderá su pedido
Luego, comienza a ejecutarse la rutina de atención de la interrupción propiamente dicha.
detectarse: como queremos terminar de ejecutar la instrucción en curso y luego atender la interrupción, la detectamos en la etapa de Fetch
atenderse: luego de terminar la instrucción en curso y ejecutar los pasos previos a la interrupción ↖

c) ¿Dónde se almacenan las micro-instrucciones?

Las micro-instrucciones son almacenadas en una memoria destinada para tal fin, que forma parte del componente UC. Esta memoria contiene palabras de 32 bits y direcciones de 9 bits. Cada uno de los 32 bits de la palabra corresponde a una señal para algún componente.

d) Explicar cómo podría hallar el micro-programa correspondiente a una instrucción cualquiera.

La estructura de la memoria de micro-instrucciones es la siguiente:

Dirección	Inst.	Dirección	Inst.	Dirección	Inst.	Dirección	Inst.
00000xxxx	fetch	01000xxxx	MOV	10000xxxx	STR	11000xxxx	INC
00001xxxx	ADD	01001xxxx	-	10001xxxx	LOAD	11001xxxx	DEC
00010xxxx	ADC	01010xxxx	-	10010xxxx	STR*	11010xxxx	SHR
00011xxxx	SUB	01011xxxx	-	10011xxxx	LOAD*	11011xxxx	SHL
00100xxxx	AND	01100xxxx	-	10100xxxx	JMP	11100xxxx	-
00101xxxx	OR	01101xxxx	-	10101xxxx	JC	11101xxxx	-
00110xxxx	XOR	01110xxxx	-	10110xxxx	JZ	11110xxxx	-
00111xxxx	CMP	01111xxxx	-	10111xxxx	JN	11111xxxx	SET

usando el código de operación el microPC busca en el código cómo ejecutar esa instrucción

e) El taller pasado analizamos algunos micro-programas con micro-instrucciones condicionales (“IF”). ¿Qué mecanismo implementa la máquina para decidir si ejecutar su rama o no?

La máquina decide “si ejecutar la rama” comparando los valores que toman los flags y saltando a la etiqueta que se quiera ejecutar.

(2) Interrupt Controller

a) Desarrollar el componente teniendo en cuenta lo siguiente:

El flag I debe encenderse al recibir la señal STI.

El flag I debe apagarse al recibir la señal CLI.

Al recibir la señal INT se debe almacenar la interrupción de manera asincrónica.

Al recibir la señal INTA se debe dejar de almacenar la interrupción.

La salida INTR debe valer 1 si y sólo si I=1 y hubo un pedido de interrupción.

Como las entradas STI, CLI no estarán nunca las dos en 1. Y con INT, INTA pasa lo mismo. Podemos usar dos latch JK ya que como muestra su tabla de verdad sigue la lógica que estamos buscando si cuando STI se encienda, I se prenderá, cuando CLI lo hagan, I se apagará, donde I=Q

Cuando prendemos INT se almacena en el latch y

Latch JK:

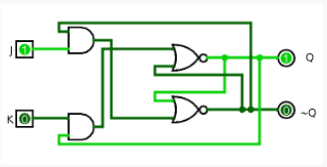

Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	$\overline{Q}*$	$Q*$

b) Dentro de la Unidad de Control ¿qué sucede si el cable con una X siempre vale 0? ¿Y si siempre vale 1?

Si el cable siempre vale 0 entonces nunca se atendería una interrupción.

Si siempre vale 1, sólo se atenderán interrupciones pero no se podrá ejecutarlas ya que para ejecutar el programa se necesita apagar ese cable para evitar que el procesador vuelva a interrumpirse.

(3) Micro-instrucciones - Agregar en el archivo microCode.ops las micro-instrucciones necesarias para que:

a) Al momento de realizar un Fetch, en caso de haber una interrupción a atender, se sigan los siguientes pasos:

1) guardar el valor de PC en la dirección 0xFF.
2) inhabilitar las interrupciones. (CLI)
3) informar que ha sido recibida la interrupción. (INTA)
4) saltar a la rutina de atención de interrupción. (RAI)

b) Las instrucciones STI y CLI manejen a I como corresponde.

c) La instrucción IRET habilite las interrupciones y salte a la dirección almacenada en la posición 0xFF.

Responder nuevamente a las preguntas del punto 2b).

2b)

(4) Ensamblar y correr - Escribir en un archivo, compilar y cargar el siguiente programa:

SET R1, 0x03 ; el registro R1 guarde el valor 0x03
SET R2, 0x00 ; R2 guarde el valor 0x00
SET R3, rai ; R3 la dirección a la rutina de atención a la interrupción
STR [0x00], R3 ; guardar en la posición de memoria 0x00 el valor del registro R3
STI ; se prende el flag I para habilitar recibir interrupciones

loop:
CMP R1, R2 ; modifica los flags según los valores que tome R1-R2
JZ fin ; como R2 es cero, la resta anterior no va a ser 0 por lo que no se ejecuta el salto
JMP loop ; se salta a la etiqueta loop y como los valores en los registros no cambian entramos en un loop

fin:
CLI ; deshabilita la llegada de interrupciones

halt:
JMP halt ; salta a la etiqueta halt pero esto nunca va a suceder por el loop infinito

rai:
DEC R1 ; decrementa el valor en R1
IRET ; vuelve a la instrucción que debería ejecutarse luego de ejecutar la interrupción

a) Antes de correr el programa, identificar el comportamiento esperado.

b) ¿Cuántas interrupciones son necesarias como mínimo para llegar a halt?
para llegar a halt debemos cumplir la condición para salir del ciclo que es que R1-R2==0. R2 es 0 y R3 es 3, por lo tanto deberíamos decrementar su valor hasta que llegue al de R2 (0) y así poder seguir la ejecución del programa. Como el valor de R1 es 3, con 3 interrupciones que decrementen el valor de R1 podemos continuar con el resto de las etiquetas.

c) ¿Qué sucede si se reciben varias interrupciones antes del CMP?
Si se recibe una interrupción antes de CMP y antes de STI no sucederá nada ya que la instrucción STI no está habilitada.
Si se recibe una interrupción antes de CMP pero luego de STI la interrupción será atendida.
Si se reciben varias, en el primer caso no serán atendidas y en el segundo caso se atenderán decrementando el valor de R1 (si son más de 3 R1 pasaría a ser un número negativo por lo que luego caeríamos en el loop infinito)

d) ¿Hay algún problema en recibir una interrupción entre el CMP y el JZ?
No, si se recibe una interrupción luego de restar R1-R2 y el valor es 0 pero se recibe una interrupción podría pasar que se decremente R1 hasta -1 pero como la operación DEC no modifica los flags Z seguiría valiendo y se continuaría deshabilitando las interrupciones y saltando a la etiqueta halt

(5) Programar

Una empresa ferroviaria adquirió un procesador OrgaSmallI para controlar sus trenes de forma segura. Con el fin de disminuir la velocidad en las curvas y así evitar accidentes, se dispone de un sensor que solicita una interrupción al detectar que se acerca una curva y otra cuando el tren termina de atravesarla.

Además de dicho sensor, se dispone de un dispositivo Motor que lee la **dirección 0xF0** periódicamente (polling) y, dependiendo del valor obtenido, realiza distintas acciones:
0x0C Cambia a velocidad para curva.
0x0F Cambia a velocidad máxima.
Para tranquilidad de los altos mandos de la empresa, se nos pide que dichos valores se escriban constantemente en la dirección mencionada.

Por último, nos piden que manejemos la bocina del tren, cuya intensidad controlamos escribiendo en la **dirección 0xF1**. Se nos pide que comience con el valor 2, y cada 32 curvas atravesadas con éxito, incrementemos su intensidad en 1 para compartir nuestra alegría con el pueblo.

Escribir un programa que maneje el tren como corresponde.

```
main:     SET R0, 0x20 ;      curvas atravesadas
          SET R1, 0x0F ;      velocidad
          STR [0xF0], R1
          SET R2, 0x02 ;      intensidad de la bocina
          STR [0xF1], R2

loop:          STR [0xF0], R1
               JMP loop

intensidad:    LOAD R2, [0xF1]
               INC R2
               STR [0xF1], R2
               IRET

rai:      CMP R1, 0x0C
          JZ velocidadFinCurva
          JMP velocidadInicioCurva


velocidadFinCurva:       SET R1, 0x0F
                         STR [0xF0], R1
                         DEC R0
                         CMP R0, 0x00
                         JZ intensidad
                         IRET

velocidadInicioCurva:    SET R1, 0x0C
                         STR [0xF0], R1
                         IRET
```