1. Ladrichuleta para micro

1. Al final del programa va la directiva `END`. Si va seguida de una etiqueta, la ejecución comienza en dicha etiqueta (T2.5, p.32)
2. Si la función se llama desde C tiene que estar declarada con `_nombre`.
3. Si el modelo de compilación de C es largo, entonces se utiliza `FAR` para procedimientos (la dirección de retorno son 4 bytes) y los punternos son de 4 bytes. Si el modelo es corto entonces se utiliza `NEAR` (la dirección de retorno son 2 bytes) y los punteros son de 2 bytes.
4. Si un procedimiento de C devuelve una palabra lo hace en `AX`. Si devuelve dos palabras, lo hace en `DX:AX`.
5. Todo procedimiento debe no tener efectos colaterales (los registros no pueden modificarse excepto en un procedimiento de C que no devuelva `void`).
6. Hay que tener en cuenta el signo de los enteros al elegir las instrucciones JMP.
7. Utilizar `CMP` en vez de `SUB` y `TEST` en vez de `AND` para las condiciones para saltar.
8. No utilizar ni `CMP` ni `SUB` si se va a comparar con `0` o con el signo y si sobre lo que se va a comparar ha sido utilizado en la última operación aritmética o lógica (las flags se mantienen para que las use la instrcción de salto).
9. Si se quiere utilizar solo un byte de memoria direccionando por registro hay que hacer un cast con `BYTE PTR` porque por defecto coge una palabra.
10. Si se accede a una variable en memoria no es necesario casting porque el tipo se define en la declaración (`VARIABLE db/dw`).
11. Si un procedimiento llama a otros y pasa argumentos por la pila haciendo push suele ser necesario hacer `add sp, N` (N entero) para limpiar la pila antes de retornar en el primer procedimiento. No se pueden utilizar `pop`s porque contaminarían los registros.
12. Los `jmp` a etiquetas que están justo debajo no hacen falta (basta con dejar que siga).
13. Comprobar los `INC`s y `DEC`s que suelen estar cambiados.
14. Al leer un `char` de la pila que hemos pasado por parámetros descartamos la parte más significativa (que deberían ser ceros y está en la dirección más baja por ser little endian).

1.1. Contenido del final: queco

1. Un ejercicio del primer parcial de obtener el valor de registros a partir de memoria e instrucciones.
2. Un problema de paso de parámetros desde C que requiere identificar los valores de la dir de retorno y demás parámetros en la pila.
3. Va a hacer un programa en su casa y luego va a quitar instrucciones para que las rellenemos (rollo segundo parcial). Va a ser perfume Yves Saint Laurent no agua de colonia.
4. Programación con entrada salida (pero puede haber de todo lo demás también).

1.2. Contenido del final: el comañero del queco

1. Hay enunciados donde pone un esquemático eléctrico. Seis páginas de enunciado. La dificultad es entender el enunciado. Una vez entendido, el problema es trivial.

1.3. Recetas para la interacción con periféricos

1.3.1. ¿Qué EOI(s) tengo que mandar para la interrupción XXh?

Ejercicios 3.3, 3.12, 3.22.
Dende del número de interrupción:

- `08h - 0Fh`: la interrupción está asociada al PIC maestro (PIC 0). Hay que mandar el EOI al PIC maestro que se encuentra en el puerto `20h`.
- `70h - 7fh`: la interrupción está asociada al PIC esclavo (PIC 1). Hay que mandar el EOI tanto a maestro como a esclavo (puerto `0A0h`).
- En otro caso: la interrupción no está asociada a ningún PIC. No hay que mandar EOIs.

Ver 5.5 § XVI y XVII para más detalles y una tabla con los periféricos asociados a las interrupciones que pasan por PICs.
¿EOI específico o no específico? TL;DR: no específico. Explicación: el EOI específico indica cómo se debe actualizar la tabla de prioridad de interrupciones en los PICs. Salvo que se diga lo contrario (no se dice en las diapositivas) se manda el EOI no específico que es el `20h` (sí, coincide con el puerto del PIC maestro pero son cosas distintas).
Nota: el EOI se manda escribiendo en el puerto correspondiente: `MOV AL, 20h, OUT 0A0h, AL`.

1.3.2. RTC = Real Time Clock

El RTC dispone de 64 bytes de memoria de los cuales 4 son de control. Se organizan en 4 registros (`0Ah, 0Bh, 0Ch, 0Dh`) que configuran distintas funcionalidades (Ver 6.3 § III-VIII).

Lectura y escritura en los registros de control Ver 6.3 § II.

1. Escribir (`OUT`) en el puerto 70h la dirección del registro de control al que accederemos (guardando dicha dirección en `AL`).
2. Acceder al registro (`IN/OUT`) a través del puerto 71h y utilizando `AL` de nuevo para la escritura/lectura de datos.

Programación del RTC para que genere interrupciones periódicas a f Hz Ejercicios 3.5, 3.18.
Nota: no siempre es posible obtener interrupciones a f Hz pero podemos aproximarnos bastante. Además, f no puede ser menor que 2 Hz por lo que si necesitamos menos frecuenciatendremos que programar el RTC a 2Hz, contar las interrupciones recibidas y actuar solamente cada $2f$ interrupciones.

1. Comprobar si se puede configurar el RTC: asegurarse de que el bit más significativo del registro A no está a 1. Ver 6.3 § IV.
2. Calcular los valores `DV` y `RS` para configurar el registro `A`.
   - `DV` es el número de reloj con la frecuencia inmediatamente superior a la f deseada
   - `RS` se obtiene de redondear $1 + \log_2 \frac{CLK}{f}$. Es necesario comprobar que dicho número cabe en 4 bits (nosotros al programarlo, no mediante ensamblador). Si no cabe porque es muy pequeño, hay que elegir un reloj más rápido. Si no, uno más lento.
3. Activar la generación de interrupciones periódicas modificando el bit 6 (`PIE`) del registro B.

Un ejemplo:

```
MOV AL, 0AH
OUT 70H, AL
IN AL, 71H
TEST AL, 10000000B
JNZ NO_SE_PUEDE_CONFIGURAR

MOV AL, 00101110B ; EL REGISTRO B DEL RTC
OUT 71H, AL

MOV AL, 0BH
OUT 70H, AL
IN AL, 71H ; LEER REGISTRO B
OR AL, 01000000B ; MODIFICAR BIT PIE
OUT 71H, AL
```

1.3.3. Rutina de interrupción vinculada al RTC (70h)

Ejercicios 3.11, 3.20, **3.21**, 3.25.
Hay que hacer 4 cosas: la primera y la última como siempre pero las dos centrales son específicas del RTC en el primer caso y de los periféricos que van por PIC en el segundo.

1. Activar las interrupciones y guardar registros.
2. Leer el registro `C` del RTC para restearlo y que se generen las siguientes interrupciones (Ver 6.3 § VIII y IX).
   - Si nos dicen que están activadas todas las interrupciones del RTC, debemos comprobar que el bit correspondiente del registro `C` está a 1. (Ver 6.3 § VIII).
3. Enviar EOIs no específicos a PIC maestro y esclavo.
4. Limpiar registros y hacer `IRET`.

Un ejemplo:

```
RSI PROC FAR
  STI
  PUSH ...

  MOV AL, 0AH
  OUT 70H, AL
  IN AL, 71H

  ; código de la interrupción

  MOV AL, 20H
```

```
  OUT 20H, AL
  OUT 0A0H, 20H

  POP ...
  IRET
RSI ENDP
```

1.3.4. Puerto paralelo (LPTX)

El puerto paralelo tiene 3 registros (datos, estado, control) que se corresponden con 3 puertos. Los números de puerto para un LPT están almacenados en un offset de memoria del segmento 0. Ver 6.5 § IV para averiguar las direcciones base.

1.3.5. Escribir en los pines 2-9 o en el registro de datos

Ejercicio 3.23.

- Los pines 2-9 del conector externo DB-25 se corresponden con los bits del registro de datos pero *en orden inverso*. Ver 6.5 § VI.
- Para leer/escribir en el registro de estado necesitamos el puerto asociado:

  1. Leer la dirección base de memoria en `DX`.
  2. Leer el registro de control desde el puerto (mediante `IN AL, DX`)
  3. Modificar lo que sea necesario en `AL` y escribir.

Un ejemplo:

```
MOV AX, 0
MOX ES, AX
MOV DX, ES:[040Ah]

IN AL, DX
; modificar AL (o escribir directamente)
OUT DX, AL
```

1.3.6. Leer/escribir en los registros de estado o control

Ejercicios 3.13, **3.21**.

- Se hace igual pero hay que tener en cuenta dos cosas

  1. Los puertos de estos registros están también en memoria pero en lugar de en la dirección base que aparece en 6.5 § IV, en `dirección base + 1` o `dirección base + 2`, respectivamente. Ver 6.5 § V.
  2. Algunos bits de estos registros están invertidos. Esto significa que **si queremos poner un 1 necesitamos escribir un 0 o viceversa. Ocurre lo mismo al leer.** Los bits que se invierten están especificados con la etiqueta `(inv)` en 6.5 § VII y VIII.