Sistema operativo y programas

domingo, 18 de octubre de 2020 15:56

SISTEMAS OPERATIVOS

Los sistemas operativos permiten admistrar el CPU para permitir la ejecución de varios programas

Cuándo yo prendo una computadora ...

- El primer programa que se ejecuta es el sistema operativo
- Al correr otros programas, el sistema operativo le da un turno a cada uno
- El SO toma y pasa el control. Cada programa tiene la ilusión de que se ejecuta solo, pero en realidad está compartiendo el tiempo del procesador

Lo mismo paso con los dispositivos ...

el sistema operativo permite abstraernos de los dispositivos y darnos alguna manera de interactuar de una manera más simple desde los programas

eso lo hace mediante los drivers. (* subprogramas específico para cada dispositivo)

* el programa le pide al SO -> el SO llama a los drivers que son los que se comunican con el dispositivo

DRIVERS

- son distintos para cada dispositivo

- son subprogramas

Como funciona INT 7

domingo, 18 de octubre de 2020 16:06

INTERRUPCIONES POR SOFTWARE

- Son similares a las subrutinas
- Programa llama al SO (mediante la interrupción)
- SO interactúa con los dispositivos (mediante los drivers)

Instrucción INT 7

- Similar a Write ("cadena") de Pascal
- Pensarlo como una subrutina
- Parámetros de entrada:
  - Bx : dirección del primer carácter de la cadena
  - Al: cantidad de caracteres de la cadena
- Parámetro de salida
  - Ninguno.

```
1  org 1000h
2  cadena db "Hola!"
3
4  org 2000h
5  mov bx , offset cadena
6  mov al , 5
7  int 7
8  int 0
9  end
```

De esa manera se imprime "Hola!" en la pantalla usando la INT 7

Se pasa en ***bx*** el inicio de la cadena

Y se pasa en ***al*** el largo de la cadena

¿Cómo se decodifica una cadena?

Dirección	Valor	Dirección	Valor
1000h	H	1000h	48h
1001h	O	1001h	4Fh
1002h	L	1002h	4Ch
1003h	A	1003h	41h
1004h	!	1004h	21h
1005h	?? (basura)	1005h	?? (basura)

Mov al , 5

Le dice a mi programa que son solo 5 caracteres, así no procesa la basura

Si no quiero , o no sé , cuantos caracteres ocupa mi cadena, lo que puedo hacer es

Mov al , offset fin - offset cadena

Cómo funciona INT 6

domingo, 18 de octubre de 2020 16:30

Instrucción INT 6

- Similar al red (variable) de Pascal
    - Solo lee caracteres
        - Lee de a uno!
- Pensarlo como una subrutina
- Parámetros de entrada
    - Ninguno
- Parámetros de salida
    - Bx : dirección dónde se almacena el carácter

¿Cómo se utiliza?

Leo un carácter

Lo almaceno en la variable "car"

Dirección	Valor
1000h	??
1001h	??

Código:

```
1  org 1000h
2  car db ?
3
4  org 2000h
5  mov bx , offset car
6  int 6
7  int 0
8  end
```

Dirección	Valor
1000h	41h
1001h	??

41h = A = ingresé A

Bien ... ¿Pero cómo leo VARIOS caracteres?

```
org 1000h
cadena db ? , ? , ? , ?
long db 5

org 2000h
 mov cl , long
 mov bx , offset cadena

loop: int 6
      inc bx
      dec cl
      jnz loop
      int 0
      end
```

Voy a tener un espacio de memoria de 5 caracteres dónde estará basura en un principio

Hago un arreglo en cadena db dónde no tengo nada, tengo basura

En el loop lo primero que hago es leer un carácter

Lo segundo es el inc bx para apuntar al siguiente carácter

Decremento a cl

Si cl llega a 0, salgo del programa

.... Si no , vuelvo al loop

¿Cómo hago para leer hasta cierto carácter? (sin longitud fija)

```
org 1000h
car_fin db "." ; creo una variable fin, con el valor de un punto.
cadena  db ? ; importante el signo de pregunta porque no reserva
; importante que la cadena esté abajo de car_fin, sino lo sobreescribo

org 2000h
      mov cl , car_fin ; luego para comparar
      mov bx , offset cadena  ; dónde arranca la cadena
loop: int 6
      cmp [bx] , cl ; acá comparo si llegó al fin
      jz fin  ; como no llegó , sigue
      inc bx ; me va a la siguiente posición de memoria
      jmp loop ; salgo al loop nuevamente

fin:  int 0
END
```

¿Cómo cuento caracteres?

```
org 1000h
car_fin db "." ; creo una variable fin, con el valor de un punto.
car_a   db "a"
cant_a  db 0
cadena  db ? ; importante el signo de pregunta porque no reserva
; importante que la cadena esté abajo de car_fin, sino lo sobreescribo

org 2000h
        mov bx , offset cadena
loop: int 6
        mov al , [bx] ; el carácter lo paso a al
        cmp al , car_fin ; lo comparo si es el último o no
        jz fin  ; si es el último salto
        ; si no, sigo:
        cmp al , car_a ; compara el carácter con la a
        jnz loop ; si no es una a , va a saltar
        inc cant_a ; al ser a, la cuenta
        jmp loop ; salta

fin:  int 0
END
```

Interrupciones por Hardware

domingo, 18 de octubre de 2020 17:52

¿Qué son las interrupciones por Hardware?

Debido a que la velocidad de los dispositivos externos es infinitamente menor (?) que la de, por ejemplo, el procesador, tiene que haber una manera para permitir la comunicación entre todos estos.

La manera tradicional es la consulta de estado o **polling**,

Otra manera, son las **interrupciones por hardware**

El polling

Pregunta constantemente si el dispositivo está ocupado y si lo está, espera

En este caso, la CPU va a estar esperando un montón de tiempo a un dispositivo mucho más lento

Interrupciones por hardware

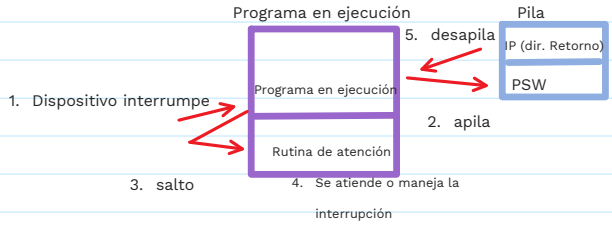
La CPU ejecuta un programa de manera normal (como veníamos hasta ahora) sin preocuparse por la interrupción de los dispositivos.

Entonces en algún momento alguien aprieta una tecla que genera una señal, que le dice a la CPU "Espera, te quiero interrumpir"

En ese momento la CPU para de ejecutar el programa y ejecuta la subrutina para atender esta interrupción (*manejador_int*)

Mecanismo de interrupción

1. Dispositivo interrumpe al programa en ejecución
2. Se apila el PSW (en el caso del simulador, los flags, pero es toda la información necesaria para regresar al programa) ya que no sé cuándo me va a interrumpir la interrupción (que me puede cambiar los flags)
3. Salta a la subrutina, como si fuese un *call*
4. Se maneja o atiende la interrupción
5. Se desapila y regresa al programa principal

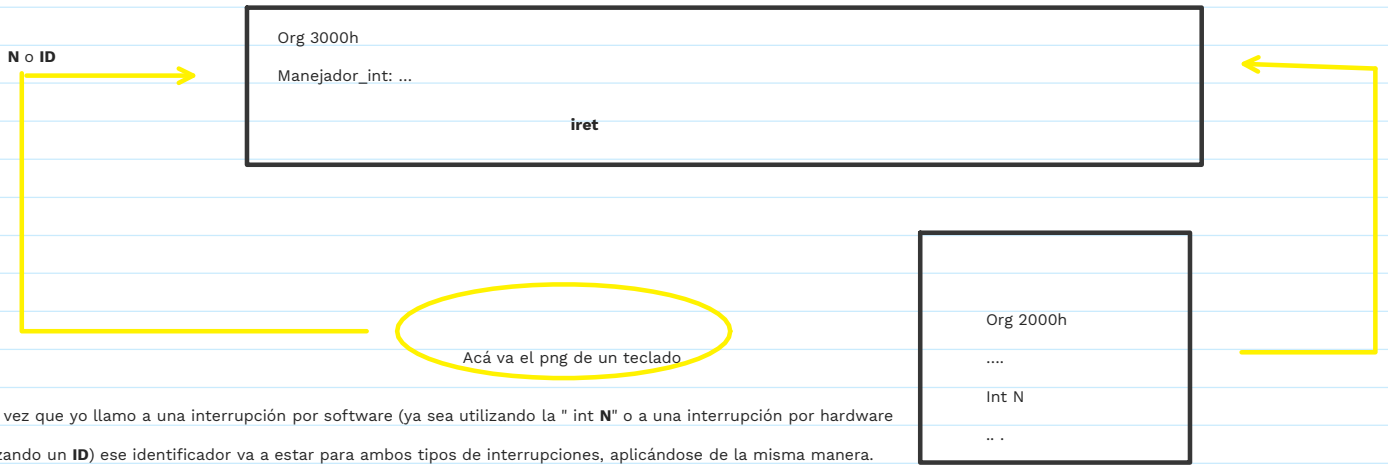


Interrupciones por software

- INT 0 (HLT)
- INT 6 (leo un carácter)
  - BX tiene la dirección dónde está almacenado dicho carácter
- INT 7 (imprime un string)
  - BX tiene la dirección dónde comienza dicho string
  - AL contiene la cantidad de caracteres a imprimir

¿Y qué tienen que ver?

Bueno, las interrupciones por software utilizan el mismo mecanismo que las interrupciones por hardaware. Usan un número para saber que tipo de subrutina ejecutar y tiene que ver con esto:



Cada vez que yo llamo a una interrupción por software (ya sea utilizando la " int **N**" o a una interrupción por hardware (utilizando un **ID**) ese identificador va a estar para ambos tipos de interrupciones, aplicándose de la misma manera.

PIC - Dispositivo interno

domingo, 18 de octubre de 2020 18:23

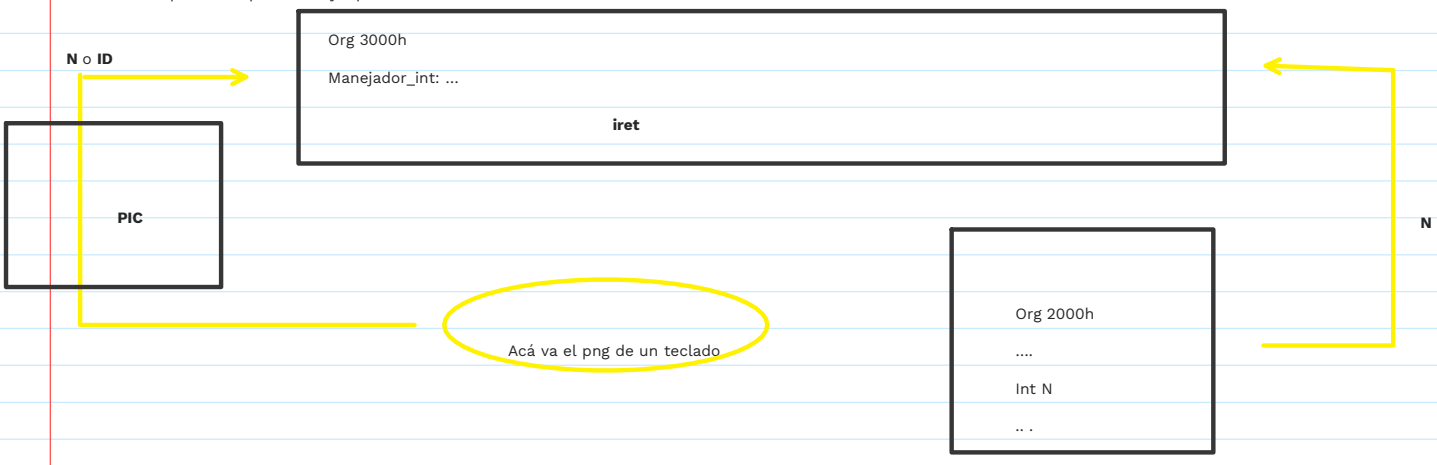
¿Cómo interrumpir con varios dispositivos de E/S?

La CPU tan solo tiene una sola línea de interrupción, por lo cuál se usa el PIC

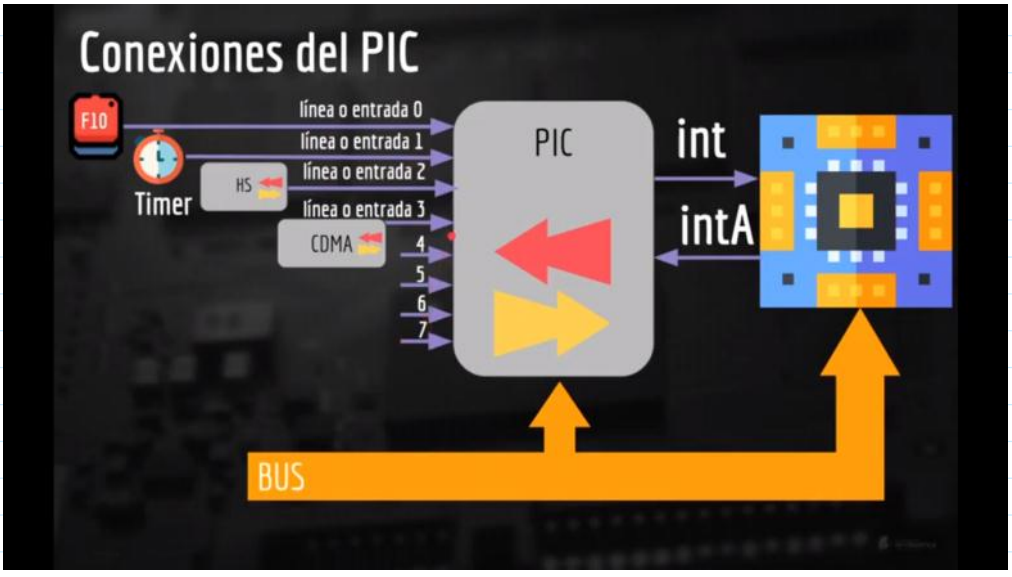
El PIC puede tener conectado varios dispositivos

Dispositivo interno: PIC

- Es el "secretario" de la CPU
- Permite conectar varios dispositivos a la vez
- Decide qué interrupciones dejar pasar



Las conexiones del PIC



Sólo va a tener 8 líneas de entradas, de las cuáles solo vamos a utilizar 4.

En la entrada 0 está el F10

En la línea 1 está el timer

En la línea 2 está el hand-shake

En la línea 3 va estar el CDMA

La 4 , 5 , 6 y 7 están libres.

El PIC se comunica con la CPU gracias a la línea **int** y la CPU se conecta con el PIC con la línea **intA**.

Además el PIC y la CPU se conecta mediante los buses para intercambiar un poco más de información.

PIC - Registros internos

EOI		20H
IMR		21H
IRR		22H
ISR		23H
INT0		24H
INT1		25H
INT2		26H
INT3		27H
INT4		28H
INT5		29H
INT6		30H

EOI: para finalizar una interrupción

IMR: interrupciones habilitadas

IRR: interrupciones pedidas

ISR: interrupciones en ejecución

INT0 a INT7: ID de interrupciones de cada dispositivo

IMR/IRR/ISR

Son registros de estados / configuraciones

- 8 bits
- 1 bit por conexión

- **IMR:** Interrupciones habilitadas
  - **I**nterruprion **M**ask **R**egister
  - 0 -> habilitada , 1 -> deshabilitada

IMR = 0FFh -> 11111111

- Todas las interrupciones deshabilitadas

= 0FDh -> 1111 1101

- El timer está habilitado

INT5		29H
INT6		30H
INT7		31H

- **I**nterruprion **M**ask **R**egister
- 0 -> habilitada , 1 -> deshabilitada
- 0 -> habilitada , 1 -> **enmáscarada**

- **IRR:** Interrupciones pedidas
  - **I**nterrupt **R**equest **R**egister
  - 0 -> **no** pedida , 1 -> pedida
- **ISR:** Interrupciones en ejecución
  - **I**nterrupt **I**n-**S**ervice **R**egister
  - 0 -> **no** en ejecución , 1 -> en ejecución

- Todas las interrupciones deshabilitadas

= 0FDh -> 1111 1101

- El timer está habilitado

IRR = 0 -> 00000000

- Ningún dispositivo está esperando para interrumpir a la CPU

ISR = 0 = 0 00000000

- La CPU no está atendiendo ninguna interrupción (no está ejecutando la subrutina manejadora de ninguna interrupción)

IMR vs IRR / ISR

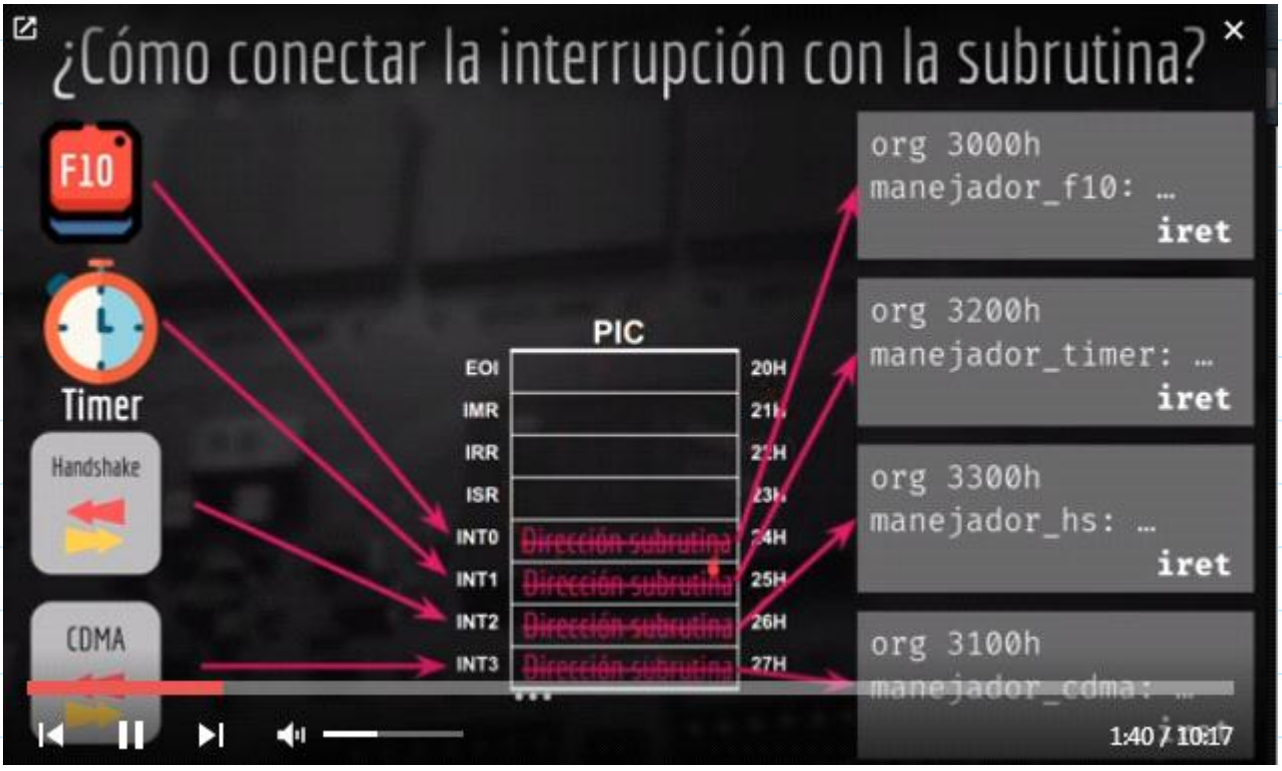
- **IMR**
  - Registro de configuración
  - Puedo *escribirlo*
    - Mov al , <máscara>
    - Out 21h , al ; 21 ah es porque ahí está el IMR
- **IRR e ISR**
  - Son dispositivos internos del pic
  - Registro de estado
  - Sólo lectura
    - No los vamos a leer
    - Sólo para ver en el simulador

Vector de interrupciones

jueves, 22 de octubre de 2020 19:04

¿Cómo es que el número de interrupción se puede asociar a la subrutina que maneja la interrupción?

Para eso un enfoque que funciona es que el PIC en los registros INT0 , INT1 , INT2 e INT3



Para independizarse del tamaño de la memoria, lo que tiene el PIC es un **identificador** de 8 bits; así que tenemos un "pasito" extra para llegar desde la interrupción hacía la subrutina que la atiende. Para esto tenemos el **VECTOR DE INTERRUPCIONES.**

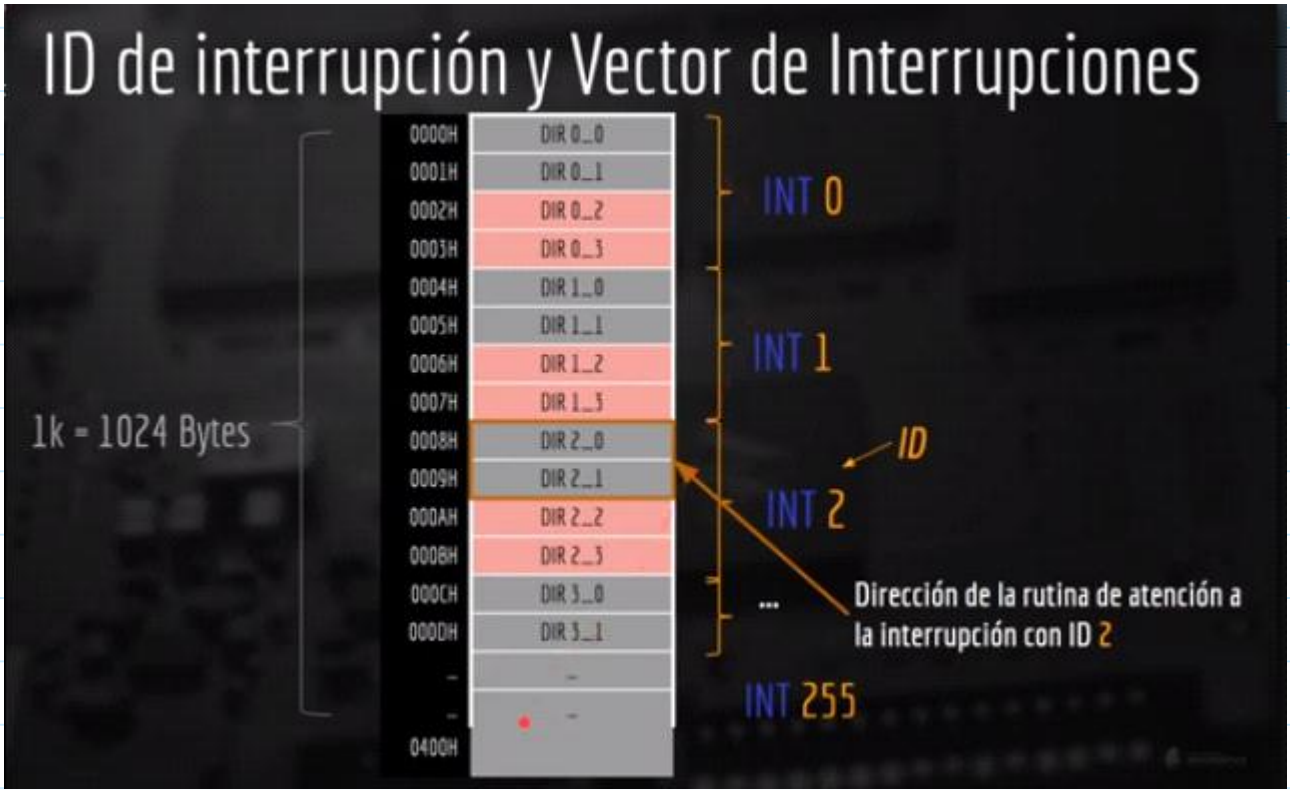
VECTOR DE INTERRUPCIONES:

Cuándo yo hago una interrupción, lo que se hace es ir al PIC y el PIC va a buscar el identificador (del 0 al 256 porque es 1B) y con el identificador de interrupción va al vector de interrupciones, dónde encontrará la dirección (codificada en el vector) dónde se encuentra la subrutina a llamar.

Es un proceso de dos pasos.

Es un número de 8bits independiente de la arquitectura de la computadora.

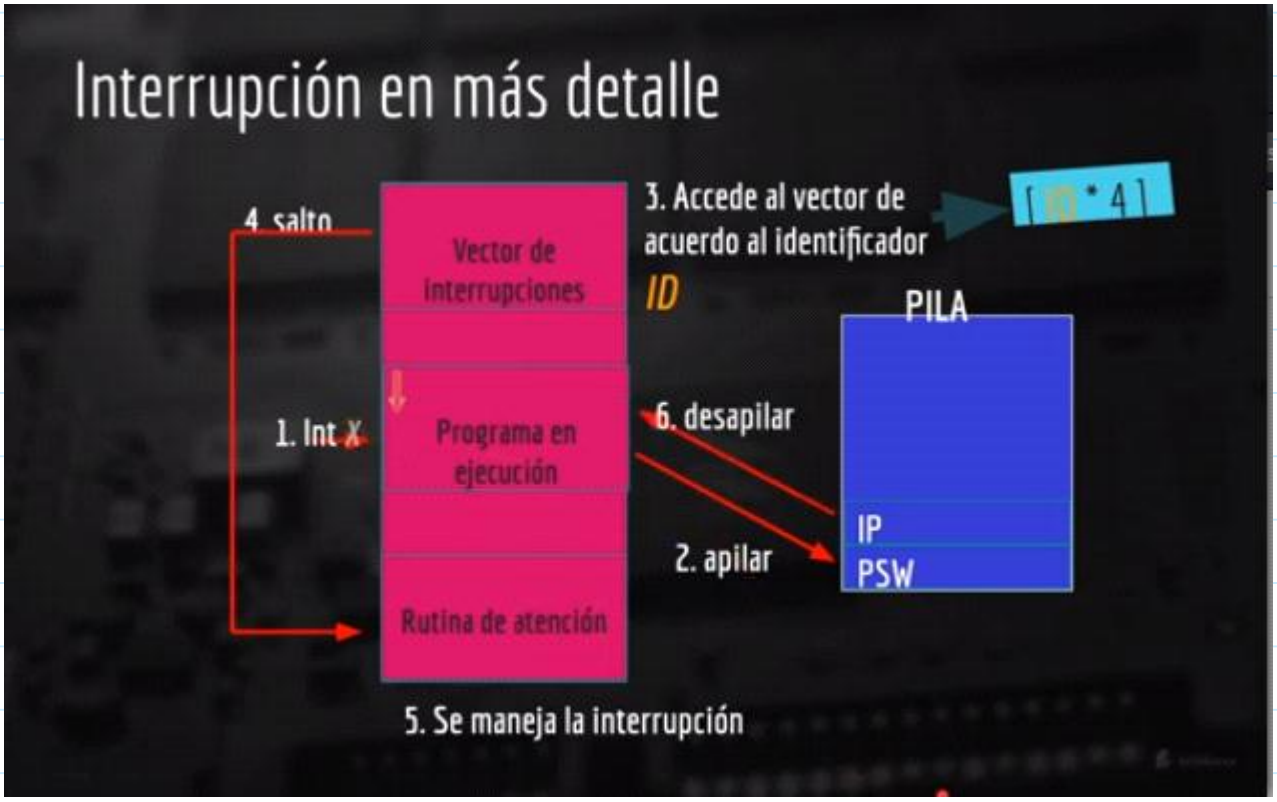
¿Qué es el vector de interrupciones?



Bueno , el vector de interrupciones no es más que un cacho de memoria que tiene un uso especial. De la misma manera que la pila que está al fin de la memoria (y tiene su uso especial) el vector de interrupciones está en el principio, ocupando los primeros 1024 B (1K -> del 0H al 400H)

- Hay 4B para codificar la dirección , por eso siempre multiplico por 4 para buscar la subrutina. Si bien solo se utilizan los dos primeros bytes, los que hicieron el procesador son precavidos y reservaron 4 ¿Por qué? Mejor prevenir que llorar en Assembly (?)
- Siempre estamos manejando tres números: el ID de la dirección (el del PIC) otro es la dirección en el vector de interrupciones (correspondiente al ID) y otro es la dirección a la subrutina.

¿Cómo funciona?



1. Tengo una interrupción X
2. Apilo el estado
3. Acceso al vector de interrupciones, con ID * 4
4. Salto a la rutina
5. Se maneja la interrupción con la subrutina de atención o manejador
6. Desapilo y vuelvo al programa.

Ejemplo

Quiero hacer una interrupción con la tecla F10.

i. En el pic configura como 10 la ID;

ii. Va a la posición 40 del vector de interrupciones;

iii. El vector de interrupciones tiene la de la subrutina.

¿Cómo elijo el ID de interrupción?

- Cualquier número entre 0 y 255
- Las interrupciones por software también utilizan el vector, así que tienen posiciones en el vector de interrupciones ocupadas.
  - ID s ocupadas:

- 0
- 6
- 7

- Libres: el resto
  - No usar la misma dos veces, lo que si.

Configuración del PIC

jueves, 22 de octubre de 2020 19:39

Pasos para configurar el PIC para una INT

1. Configurar el registro IMR. Para habilitar las interrupciones que nos interesan.
2. Configurar el registro INTX, dónde X es la línea de interrupción
   a. INT0 para el F10
   b. INT1 para el Timer
   c. ...
   d. ...
3. Implementar la subrutina de atención que termina con:
   a. Mandar el valor 20h al registro EOI (del PIC) para que el PIC se entere que terminó la interrupción.
   b. Instrucción *iret* (interrupt return)
4. Poner dirección de la subrutina en el vector de interrupciones. Muy importante.

Ejemplo:

- Escribir un programa
- Subrutina "saludar"
  - Que imprime el mensaje "Hola" en pantalla
- Cada vez que se presiona f10
  - Se llama a la subrutina

Pasos

0. Definiciones EQU

Una definición por cada registro a utilizar

- EOI EQU 20H
- IMR EQU 21H
- INT0 EQU 24H

1. Configurar el registro IMR

- F10 conectada a la línea 0
- Habilitación depende del bit 0 (menos significativo) del IMR
- IMR
  - 1 deshabilitado
  - 0 habilitado
    - 1111 1110
    - 0FEh

```
Mov al , 0FEh
Out IMR , al
```

2. Configurar el registro INT0

- F10 conectada a la línea 0
  - Registro INT0 tiene e ID de interrupciones
- Elijo un ID de forma arbiatraria
  - El 24, por ejemplo

```
Mov al , 24
Out INT0 , al
```

3. Escribir la subrutina

- Imprime "chau"
- Finaliza
  - Escribe 20h en EOI
  - Vuelve con iret
  - Lo mismo para todas las subrutinas de interrupción

4. Dirección de la subrutina en el Vector de interrupciones

- El ID de interrupciones es 24 (índice)
- Posición 24 del vector
  - Dirección 24 * 4 = 96 decimal
- Dirección de la subrutina = 3000h

```
Mov bx , 96
Mov word PTR [bx] , 3000h
```

PERO ¿QUÉ PASA SI A MITAD DE LA CONFIGURACIÓN ALGUIEN ME APRETA F10?

Instrucciones CLI y STI

- Sirven para desactivar (y activar) las interrupciones mientras se configura (o termina de) el PIC
- CLI
  - Deshabilita

```
Org 1000h
Msj db "chau"

Org 3000h (3)
Saludar: mov bx , offset msj
         Mov al , 4
         Int 7
         Mov al , 20 h
         Out eoi , al
         Iret

Org 2000h
CLI
Mov al , 0FEh (1)
Out IMR, al

Mov al , 24 (2)
Out INT0, al

Mov bx , 96 (4)
Mov WORD PTR [bx] , 3000h
STI
```

- Sirven para desactivar (y activar) las interrupciones mientras se configura (o termina de) el PIC
- CLI
  - Deshabilita
- STI
  - Habilita
- Siempre que se configura el PIC, se lo "encierra" entre ambas instrucciones
- Se deshabilita **no desde el PIC, sino desde el procesador** todas las interrupciones

```
STI

Loop: jmp loop (bucle infinito)

End
```

VERSIONES ALTERNATIVAS

Paso 4

- Declarar variables con un valor
  - Modifica la memoria
- ORG X
  - Indica dónde
- Podemos utilizarlo para escribir en el vector de interrupciones

```
Mov bx , 96
Mov WORD PTR [bx] , 3000h

Org 96
Dir_ saludar dw 3000h

Org 96
    Dw 3000h
```

```
mov bx , 96
mov WORD PTR [bx] , saludar

org 96
dir_saludar dw saludar
```

PASO 2 : CONSTANTE PARA EL ID

- El ID de interrupciones no queda claro en el código
- Utilizar una constante para indicarlo

PASO 3: PRESERVAR REGISTROS

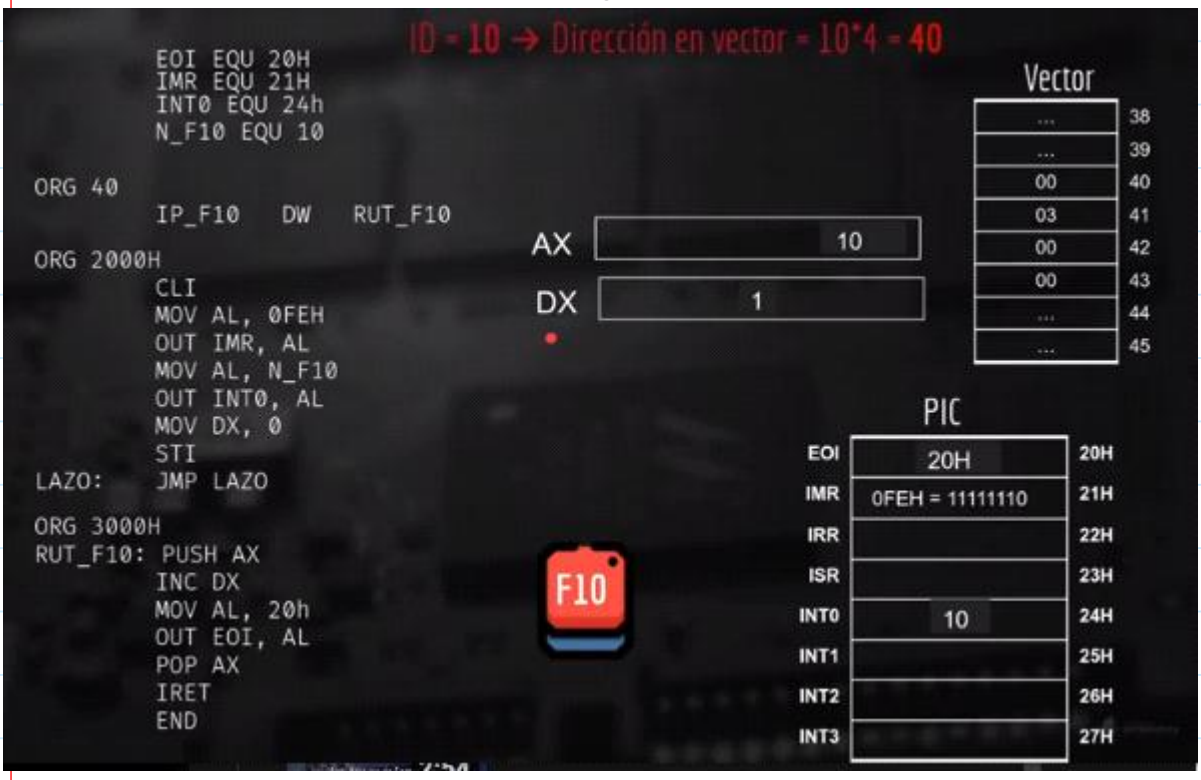
- No se sabe cuándo se va a ejecutar la interrupción
- Diseñar las subrutinas para que preserven los registros.

```
Org 3000h
Saludar: push ax
Push ax
*resto*
Pop ax
Pop bx
```

Interrupciones con la tecla F10

viernes, 23 de octubre de 2020 14:44

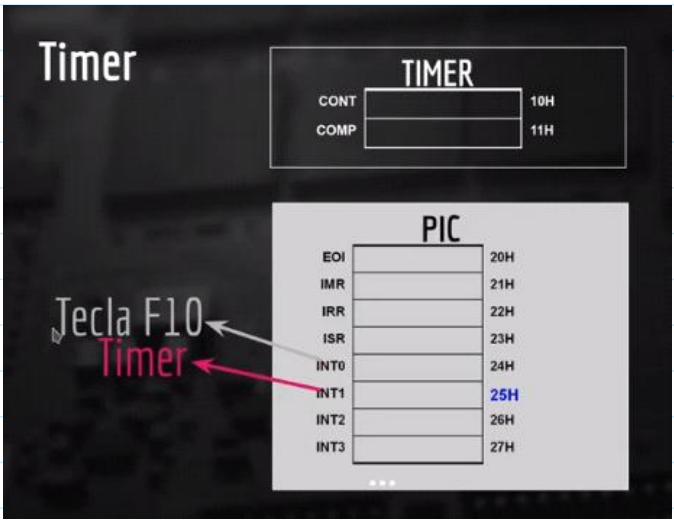
- Escribir un programa
  - Ejecutar un lazo infinito
    - Nunca termina
  - **Cuente** el número de veces que se presiona la tecla F10
    - Acumule este valor en el registro DX



Interrupciones con el Timer

viernes, 23 de octubre de 2020 14:51

Timer (temperorizador)



Posee dos registros de 8 bits

- **COMP (10h):** Registro de comparación.
- **CONT (11h):** Registro contador
  - Se incrementa una vez por segundo *automáticamente*.

Cuándo **COMP = CONT**

- BEEP, interrupción del Timer.

EJEMPLO

Implementar un programa que espere 10 segundos y , luego de esos 10 muestre un "hola"

Nota: Es MUY importante primero mandarle 0 al CONT y luego establecer el tiempo en el COMP

```
eoi equ 20h
imr equ 21h
int1 equ 25h

cont equ 10h
comp equ 11h

org 1000h
 msj db "hola!"
 flag db 0 ; me avisa que la interrupción termino

org 3100h ; subrutina de atención, va a poner el flag ese en 1
atender:  mov bx , offset msj
        mov al , 5 ; xq hola! tiene 5 carácteres
        int 7

        mov flag , 1 ; de esta manera le aviso al programa principal que ya se imprimió el mensa

        mov al , 20h ; ahora le aviso al PIC que termino
        out eoi , al

        iret

org 80 ; no h, está en decimal. acá pongo a atender
dir_atender dw atender ; 3100h

org 2000h
 cli

; configuración del PIC
 mov al , 0FDh ; 1111 1101
 out imr , al

 mov al , 20 ; id de interrupción
 out  int1 , al

;configuración del TIMER
 mov al , 0
 out cont , al

 mov al , 10
 out comp , al

 sti

loop: cmp flag , 1 ; no es un loop infinito, es un loop que compara flag con 1
      jnz loop
int 0
end

```

Timer con reinicio

viernes, 23 de octubre de 2020 15:25

- Escribir un programa que muestre en pantalla la cadena de caracteres "No debo programar en assembler" repetidamente cada 5 segundos
- El programa nunca termina

¿Cuál es el truco?

REINICIAR

- Iniciar **CONT = 0**
- Asignar **COMP = 5**
- Cada vez que **CONT = COMP** ...
    - Reiniciar **cont = 0**

```
Org 3000h
Manejador: mov al , 0
           Out CONT , al
           ...
           Iret
```

...

```
EOI equ 20h
imr equ 21h
int1 equ 25h

cont equ 10h
comp equ 11h

org 1000h
mensaje db "No debo programar en Assembly"
fin db ?

org 3000h
manejador: mov al , 0
           out cont , al

           ; ejecuta cada 5 segundos
           mov bx , offset mensaje
           mov al , offset fin - offset mensaje ; para calcular cantidad
           int 7

           mov al , 20h
           mov EOI , al ; el pic se entera que terminó la interrup.
           iret

org 32 ; 8 * 4 = 32
dir_manejador dw 3000h

org 2000h
cli
; configuramos el PIC
 mov al , 0FDh;  1111 1101
 out imr , al

 mov al , 8
 out int1 , al
; conectamos al PIC con la interrupción en 32
; configuramos el timer
 mov al , 0
 out cont , al

 mov al , 5 ; xq es una vez cada 5 segundos
 out comp , al
sti

loop: jmp loop
end
```

ES OUT EOI , AL !!!!!!!!!!!!!!!!!!!!!!!!

Registro CONT

viernes, 23 de octubre de 2020 16:22

- Se incrementa una vez por segundo
- Es de 8 bits
    - Puede contar hasta 255
    - ¿Qué sucede después?
        - Vuelve a 0

¿Qué pasa con CONT cuándo CONT = COMP?

- Se dispara una interrupción
- ¿Valor de CONT?
    - No vuelve a 0
    - Sigue incrementándose

Timer y F10

viernes, 23 de octubre de 2020 15:47

Escribir un programa que muestre en pantalla la cadena de caracteres "No debo programar en Assembly" repetidamente cada 5 segundos

El programar termina cuándo se presiona la tecla F10

¿Cómo vamos a hacer esto? : Finalizar con FLAG

- Inicializamos el FLAG en 0
- El programa principal termina cuándo FLAG = 1
- Para eso, el manejador de instrucciones cuándo presiono F10 tiene que
    - Poner el flag en 1

```
; cONSTANTES
EOI equ 20h
imr equ 21h
int0 equ 24h
int1 equ 25h

cont equ 10h
comp equ 11h

; VARIABLES
org 1000h
mensaje db "No debo programar en Assembly"
fin db ?
flag db 0

; RUTINAS
org 3200h
manejadorF10: mov flag , 1 ; indica al programa principal que deje de ejecutarse
; y vuelva a INT 0
            mov al , 20h
            out EOI  , al

            iret

org 3000h
manejador: mov al , 0
          out cont , al

          ; ejecuta cada 5 segundos
```

```
        mov bx , offset mensaje
        mov al , offset fin - offset mensaje ; para calcular cantidad
        int 7

        mov al , 20h
        out  EOI , al
        iret

; VECTOR DE INTERRUPCIONES
org 32 ; 8 * 4 = 32
dir_manejador dw 3000h

org 60 ; 15 * 4
dir_manejadorF10 dw 3200h

; PROGRAMA PRINCIPAL
org 2000h
cli
; configuramos el PIC
; confg. el IMR para el timer y F10
 mov al , 0FCh;  1111 1100
 out imr , al

 mov al , 15
 out int0 , al

 mov al , 8
 out int1 , al
; conectamos al PIC con la interrupción en 32
; configuramos el timer
 mov al , 0
 out cont , al

 mov al , 5 ; xq es una vez cada 5 segundos
 out comp , al
sti

loop: cmp flag , 1
     jnz loop

int 0
```

```
end
```