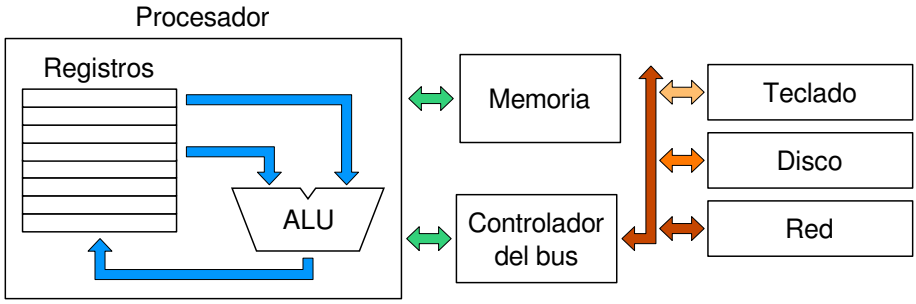
Hardware de Entrada-Salida

David Alejandro González Márquez

Clase disponible en: `https://github.com/fokerman/computingSystemsCourse`

Subsistema de Entrada-Salida

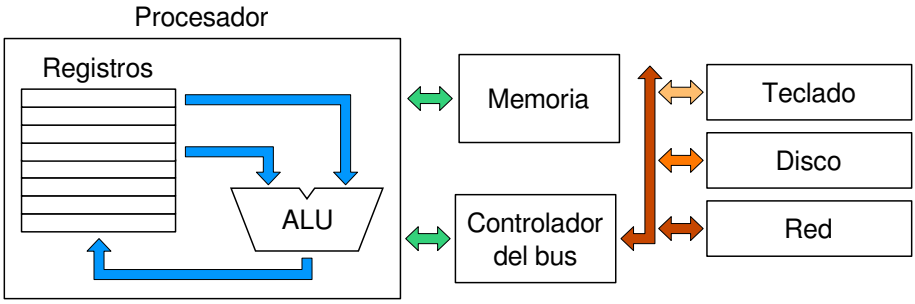
Queremos interactuar con el sistema, enviando estímulos y recibiendo respuestas.



Necesitamos **conectar** dispositivos con el procesador.

Subsistema de Entrada-Salida

Queremos interactuar con el sistema, enviando estímulos y recibiendo respuestas.



Necesitamos **conectar** dispositivos con el procesador.

Bus

Canal encargado de transferir datos entre componentes o dispositivos. Se compone por un conjunto de cables o líneas que se utilizan respetando un protocolo de control.

Buses

Los buses nos permiten conectar dispositivos entre sí.

Clasifican sus señales en tres tipos básicos:

- **Direcciones**: Conjunto de señales que conforman la dirección a leer o escribir.
- **Datos**: Conjunto de señales donde se escriben los datos a operar.
- **Control**: Señales que identifican la operación a realizar -lectura o escritura- y sus características.

Buses

Los buses nos permiten conectar dispositivos entre sí.

Clasifican sus señales en tres tipos básicos:

- **Direcciones**: Conjunto de señales que conforman la dirección a leer o escribir.
- **Datos**: Conjunto de señales donde se escriben los datos a operar.
- **Control**: Señales que identifican la operación a realizar -lectura o escritura- y sus características.

Para que múltiples dispositivos puedan usar estas líneas,
estos deben respetar un **protocolo de bus**.

El protocolo describe el conjunto de reglas para:

- Iniciar y realizar una operación sobre el bus (**master**).
- Detectar cuándo se debe responder a un pedido del bus (**slave**).

Tipos de buses

Los protocolos los podemos clasificar en:

- **Sincrónicos**: Existe una señal distinguida (clock) que sincroniza todas las etapas del protocolo del bus.
- **Asincrónicos**: No existe ninguna señal que marque el tiempo. El protocolo opera identificando cambios de estado de las señales.

Tipos de buses

Los protocolos los podemos clasificar en:

- **Sincrónicos**: Existe una señal distinguida (clock) que sincroniza todas las etapas del protocolo del bus.
- **Asincrónicos**: No existe ninguna señal que marque el tiempo. El protocolo opera identificando cambios de estado de las señales.

Según el uso de las líneas:

- **Dedicadas**: Cada línea tiene un propósito específico.
- **Multiplexadas**: Cada línea varía su uso de acuerdo a en qué etapa se encuentra del protocolo.

Tipos de buses

Los protocolos los podemos clasificar en:

- **Sincrónicos**: Existe una señal distinguida (clock) que sincroniza todas las etapas del protocolo del bus.
- **Asincrónicos**: No existe ninguna señal que marque el tiempo. El protocolo opera identificando cambios de estado de las señales.

Según el uso de las líneas:

- **Dedicadas**: Cada línea tiene un propósito específico.
- **Multiplexadas**: Cada línea varía su uso de acuerdo a en qué etapa se encuentra del protocolo.

Según el modo de transferencia:

- **Serie**: La información se envía de forma secuencial, bit tras bit.
- **Paralelo**: La información se envía simultáneamente por distintas líneas.

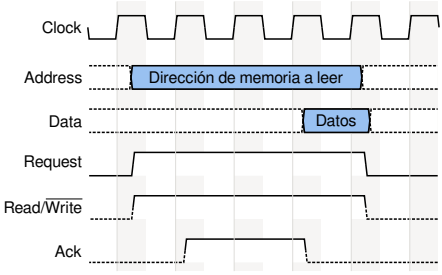
Ejemplo protocolos

Sincrónico

Asincrónico

Ejemplo protocolos

Sincrónico

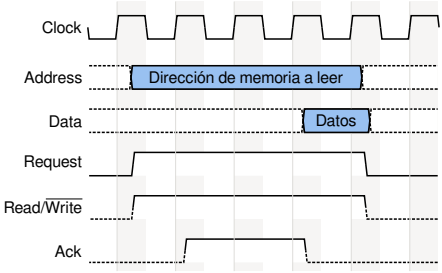


1. El `master` coloca la dirección en el bus, levanta su señal de `request` y setea la señal de lectura.
2. El `slave` levanta la señal de ack hasta que tiene el dato.
3. El `slave` baja la señal de ack y al ciclo siguiente presenta el dato durante un clock.
4. El `master` baja todas las señales.

Asincrónico

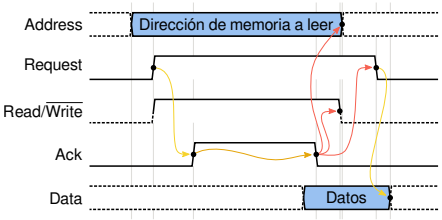
Ejemplo protocolos

Sincrónico



1. El `master` coloca la dirección en el bus, levanta su señal de `request` y setea la señal de lectura.
2. El `slave` levanta la señal de ack hasta que tiene el dato.
3. El `slave` baja la señal de ack y al ciclo siguiente presenta el dato durante un clock.
4. El `master` baja todas las señales.

Asincrónico



1. El `master` coloca la dirección en el bus, levanta su señal de `request` y setea la señal de lectura.
2. El `slave` escucha la señal de `request`, comienza la lecuta y levanta la señal de ack.
3. El `slave` presenta el dato leído y baja la señal de ack.
4. El `master` lee el dato y baja la señal de `request`.

Árbitro del bus

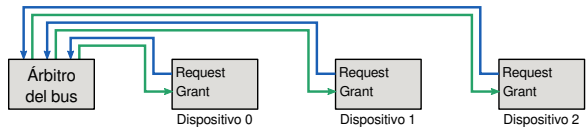
Cuando sobre un bus existen múltiples dispositivos que pueden inciar una operación, es decir, actuar como **master**, se debe arbitrar el uso del mismo.

Árbitro del bus

Cuando sobre un bus existen múltiples dispositivos que pueden inciar una operación, es decir, actuar como **master**, se debe arbitrar el uso del mismo.

Árbitro centralizado

Existe un dispositivo denominado árbitro, que recibe todas las señales de solicitud del bus y decide a qué dispositivo le otorga el uso.

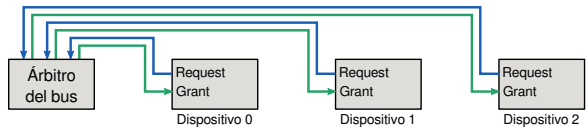


Árbitro del bus

Cuando sobre un bus existen múltiples dispositivos que pueden inciar una operación, es decir, actuar como **master**, se debe arbitrar el uso del mismo.

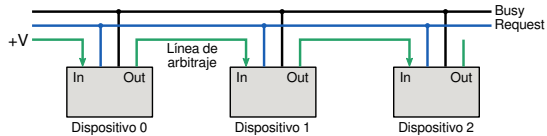
Árbitro centralizado

Existe un dispositivo denominado árbitro, que recibe todas las señales de solicitud del bus y decide a qué dispositivo le otorga el uso.



Árbitro descentralizado

Cada dispositivo recibe información parcial de las solicitudes de uso del bus y cada uno decide independiente y de forma determinística a qué dispositivo se le otorga el bus.



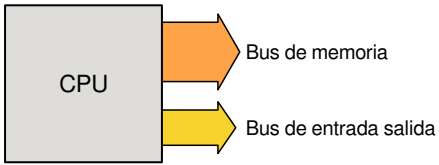
Interfaz de Entrada-Salida

Exiten dos formas básicas de acceder a dispositivos de E/S:

Interfaz de Entrada-Salida

Exiten dos formas básicas de acceder a dispositivos de E/S:

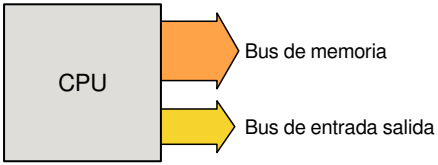
Bus de E/S independiente
El procesador cuenta con un bus especial para acceder a un espacio de entrada-salida.
Se suelen utilizar instrucciones específicas para acceder a este espacio (`in` y `out`).



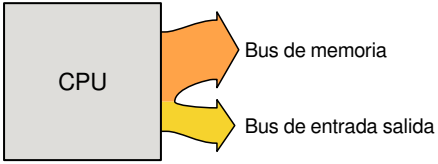
Interfaz de Entrada-Salida

Exiten dos formas básicas de acceder a dispositivos de E/S:

Bus de E/S independiente
El procesador cuenta con un bus especial para acceder a un espacio de entrada-salida.
Se suelen utilizar instrucciones específicas para acceder a este espacio (`in` y `out`).



Espacio de E/S mapeado a memoria
El controlador del bus se ocupa de decodificar las direcciones y dependiendo del rango, se accede a la memoria principal o a memoria de un dispositivo.
Se accede al espacio de la misma forma que una lectura o escritura a memoria.

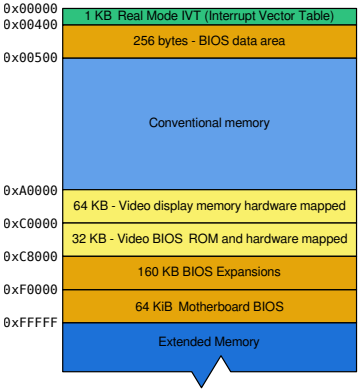


Ejemplos de Entrada-Salida

Ejemplos de Entrada-Salida

Ejemplo: Mapeado a memoria

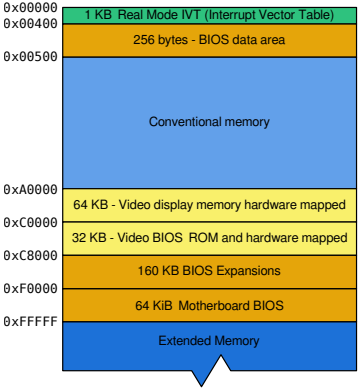
Memoria de video.



Ejemplos de Entrada-Salida

Ejemplo: Mapeado a memoria

Memoria de video.



Ejemplo: Espacio de E/S

Dispositivo controlador de teclado.

COMMANDS (I/O ADDRESS HEX 64)

COMMAND	FUNCTION
20	Read Command Byte of Keyboard Controller
60	Write Command Byte of Keyboard Controller **BIT** **BIT DEFINITIONS** 7 Reserved 6 IBM PC Compatible Mode 5 IBM PC Mode 4 Disable Keyboard 3 Inhibit Override 2 System Flag 1 Reserved 0 Enable Output Buffer Full Interrupt
AA	Self-test **BIT** **BIT DEFINITIONS** 00 No Error Detected 01 K/B Clock Line is Stuck Low 02 K/B Clock Line is Stuck High 03 K/B Data Line is Stuck Low 04 K/B Data Line is Stuck High
AB	Interface Test
AD	Disable Keyboard Feature
AE	Enable Keyboard Interface
C0	Read Input Port
D0	Read Output Port
D1	Write Output Port
E0	Read Test Inputs
F0-FF	Pulse Output Port

Eventos de Entrada-Salida

Además de interactuar, realizando lecturas y escrituras sobre el dispositivo de E/S, debemos poder detectar **eventos** que le suceden a los dispositivos de E/S.

- **Espera activa**
  Se consulta todo el tiempo el estado de los datos y se realiza una acción en base al cambio de estados (Ej. Capturadora de sonido).

Eventos de Entrada-Salida

Además de interactuar, realizando lecturas y escrituras sobre el dispositivo de E/S, debemos poder detectar **eventos** que le suceden a los dispositivos de E/S.

- **Espera activa**
  Se consulta todo el tiempo el estado de los datos y se realiza una acción en base al cambio de estados (Ej. Capturadora de sonido).

- **Interrupciones**
  El procesador se detiene, guarda el estado de actual y comienza a ejecutar la rutina que atenderá el pedido (Ej. Teclado).

Eventos de Entrada-Salida

Además de interactuar, realizando lecturas y escrituras sobre el dispositivo de E/S, debemos poder detectar **eventos** que le suceden a los dispositivos de E/S.

- **Espera activa**
  Se consulta todo el tiempo el estado de los datos y se realiza una acción en base al cambio de estados (Ej. Capturadora de sonido).

- **Interrupciones**
  El procesador se detiene, guarda el estado de actual y comienza a ejecutar la rutina que atenderá el pedido (Ej. Teclado).

- **Acceso directo a memoria (DMA)**
  Se configura un dispositivo especial, que actúa como master en el bus y permite copiar datos entre diferentes dispositivos (Ej. Copia de bloques en el disco rígido).

Ejemplos de Entrada-Salida

Suponer un sistema conectado a una capturadora de datos. La capturadora tiene dos registros: uno que indica el número de muestra (`counter`) y otro que indica el valor de la muestra (`value`). Estos están expuestos en direcciones de memoria.

Ejemplos de Entrada-Salida

Suponer un sistema conectado a una capturadora de datos. La capturadora tiene dos registros: uno que indica el número de muestra (`counter`) y otro que indica el valor de la muestra (`value`). Estos están expuestos en direcciones de memoria.

- **Espera activa**

```
current = 0
main:
    while(true):
        if current != counter:
            guardarMuestra(counter, value)
            current = counter
```

Ejemplos de Entrada-Salida

Suponer un sistema conectado a una capturadora de datos. La capturadora tiene dos registros: uno que indica el número de muestra (`counter`) y otro que indica el valor de la muestra (`value`). Estos están expuestos en direcciones de memoria.

- **Espera activa**

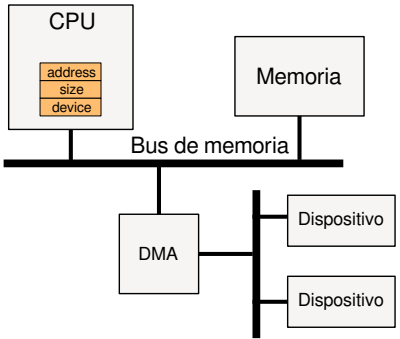
```
current = 0
main:
    while(true):
        if current != counter:
            guardarMuestra(counter, value)
            current = counter
```

- **Interrupciones**

```
int:
    guardarMuestra(counter, value)
    iret
```

DMA (direct memory access)

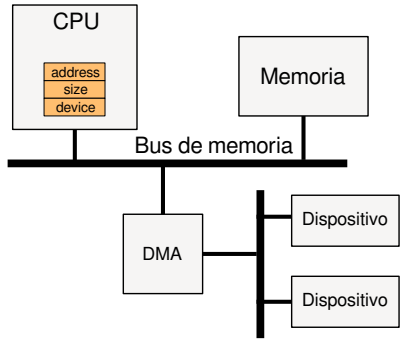
El DMA es un dispositivo programable que actúa como `master` del bus en remplazo de la CPU.



DMA (direct memory access)

El DMA es un dispositivo programable que actúa como `master` del bus en remplazo de la CPU.

La idea es que la CPU pueda seguir procesando mientras el DMA se ocupa de la copia de datos.

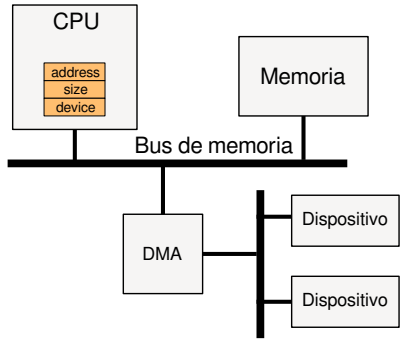


DMA (direct memory access)

El DMA es un dispositivo programable que actúa como `master` del bus en remplazo de la CPU.

La idea es que la CPU pueda seguir procesando mientras el DMA se ocupa de la copia de datos.

La CPU configura al DMA para copiar datos entre los dispositivos y la memoria.



DMA (direct memory access)

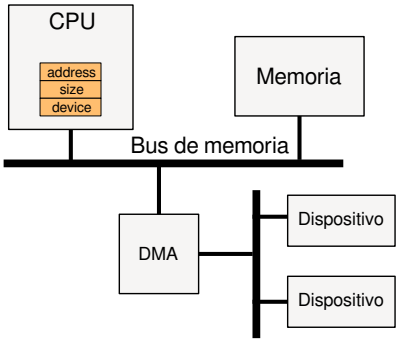
El DMA es un dispositivo programable que actúa como `master` del bus en remplazo de la CPU.

La idea es que la CPU pueda seguir procesando mientras el DMA se ocupa de la copia de datos.

La CPU configura al DMA para copiar datos entre los dispositivos y la memoria.

La configuración de un DMA consiste en:

- `address`: Dirección de comienzo del bloque de memoria a copiar.
- `size`: Cantidad de bytes a procesar.
- `device`: Identificador del dispositivo.
- `action`: Leer o escribir.



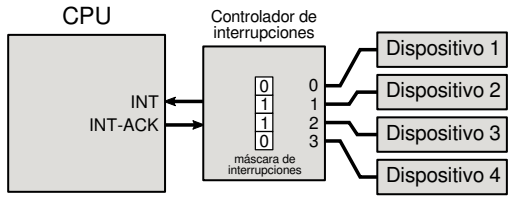
Controlador de interrupciones

Los sistemas pueden contar con una o múltiples entradas de interrupciones.

Controlador de interrupciones

Los sistemas pueden contar con una o múltiples entradas de interrupciones.

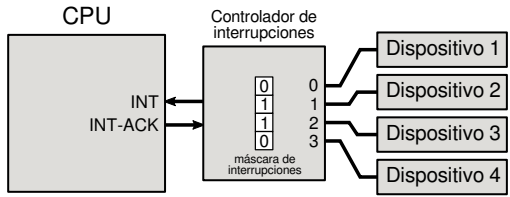
Si quisieramos tener más interrupciones, contamos con un componente nuevo: el controlador de interrupciones.



Controlador de interrupciones

Los sistemas pueden contar con una o múltiples entradas de interrupciones.

Si quisieramos tener más interrupciones, contamos con un componente nuevo: el controlador de interrupciones.



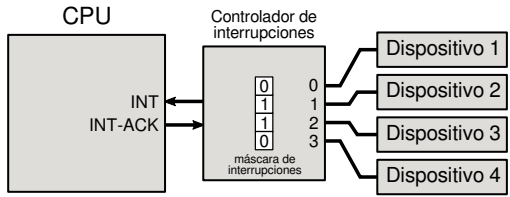
Este dispositivo se ocupa de:

- Determinar qué interrupción se produjo teniendo en cuenta su **prioridad**.
- Establecer el **vector de interrupciones**, indicando el número de interrupción producida.
- Validar si las interrupciones deben ser atendidas en base a la **máscara de interrupciones**.

Controlador de interrupciones

Los sistemas pueden contar con una o múltiples entradas de interrupciones.

Si quisieramos tener más interrupciones, contamos con un componente nuevo: el controlador de interrupciones.



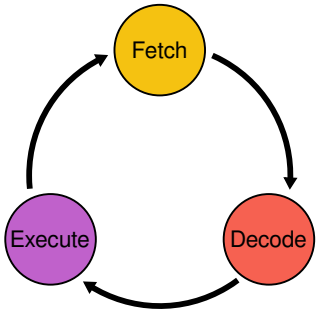
Este dispositivo se ocupa de:

- Determinar qué interrupción se produjo teniendo en cuenta su **prioridad**.
- Establecer el **vector de interrupciones**, indicando el número de interrupción producida.
- Validar si las interrupciones deben ser atendidas en base a la **máscara de interrupciones**.

La señal `INT-ACK` es producida por el procesador y le indica al controlador de interrupciones cuando una interrupción fue atendida.

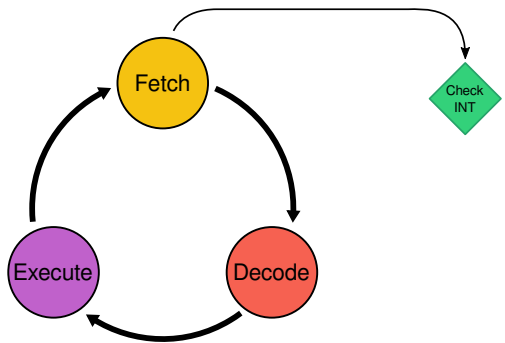
Relación de las interrupciones con el ciclo de instrucción

Las interrupciones **no se pueden atender instantáneamente** cuando llegan ya que no es posible ejecutar la mitad de una instrucción. Esto determina que el procesador decide cuándo comprobar si existen interrupciones y cuándo estas serán atendidas. En general, este proceso ocurre previo al *fetch* de la instrucción.



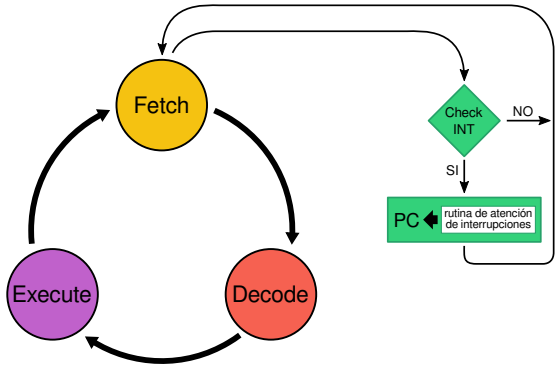
Relación de las interrupciones con el ciclo de instrucción

Las interrupciones **no se pueden atender instantáneamente** cuando llegan ya que no es posible ejecutar la mitad de una instrucción. Esto determina que el procesador decide cuándo comprobar si existen interrupciones y cuándo estas serán atendidas. En general, este proceso ocurre previo al *fetch* de la instrucción.



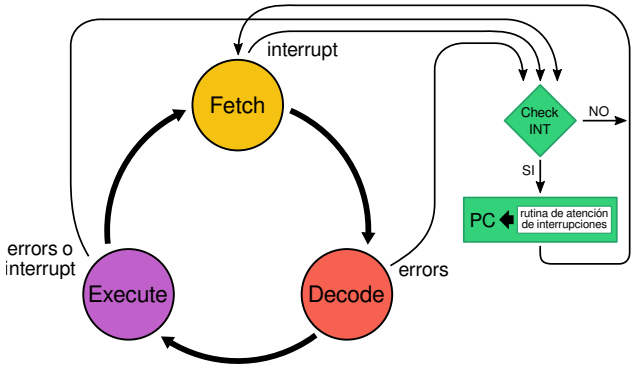
Relación de las interrupciones con el ciclo de instrucción

Las interrupciones **no se pueden atender instantáneamente** cuando llegan ya que no es posible ejecutar la mitad de una instrucción. Esto determina que el procesador decide cuándo comprobar si existen interrupciones y cuándo estas serán atendidas. En general, este proceso ocurre previo al *fetch* de la instrucción.



Relación de las interrupciones con el ciclo de instrucción

Las interrupciones **no se pueden atender instantáneamente** cuando llegan ya que no es posible ejecutar la mitad de una instrucción. Esto determina que el procesador decide cuándo comprobar si existen interrupciones y cuándo estas serán atendidas. En general, este proceso ocurre previo al *fetch* de la instrucción.



Tipos de interrupciones

Dependiendo de la fuente de generación de las interrupciones, las podemos clasificar en:

Tipos de interrupciones

Dependiendo de la fuente de generación de las interrupciones, las podemos clasificar en:

- **Interrupciones externas**: Son producidas por dispositivos externos y llegan al procesador mediante una señal enviada por el controlador de interrupciones (Ej: teclado).

Tipos de interrupciones

Dependiendo de la fuente de generación de las interrupciones, las podemos clasificar en:

- **Interrupciones externas**: Son producidas por dispositivos externos y llegan al procesador mediante una señal enviada por el controlador de interrupciones (Ej: teclado).

- **Interrupciones sincrónicas**: Son generadas mediante instrucciones que cambian el flujo del programa. Generalmente se utilizan para modelar servicios del sistema operativo. Dependiendo del procesador, se puede utilizar un mecanismo de interrupciones o instrucciones específicas para llamar a servicios del sistema. (Ej: `int 0x80`).

Tipos de interrupciones

Dependiendo de la fuente de generación de las interrupciones, las podemos clasificar en:

- **Interrupciones externas**: Son producidas por dispositivos externos y llegan al procesador mediante una señal enviada por el controlador de interrupciones (Ej: teclado).

- **Interrupciones sincrónicas**: Son generadas mediante instrucciones que cambian el flujo del programa. Generalmente se utilizan para modelar servicios del sistema operativo. Dependiendo del procesador, se puede utilizar un mecanismo de interrupciones o instrucciones específicas para llamar a servicios del sistema. (Ej: `int 0x80`).

- **Excepciones**: Es un tipo especial de interrupción que se da cuando el procesador genera un error. La interrupción detiene el programa, ejecutando código que permite entender qué sucedió y arreglarlo (Ej: Intentar ejecutar una instrucción inválida o ejecutar una división por cero).
  Cuando se produce un error, se pueden dar tres casos:
  - *volver a ejecutar la instrucción que produjo el error luego de arreglarlo* (**fault**)
  - *ejecutar la próxima instrucción porque nada se puede hacer* (**trap**)
  - *no se puede conocer exactamente dónde se produjo el error* (**abort**)

Rutinas de atención de interrupciones

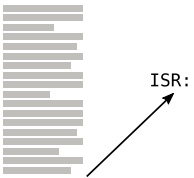
Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

Rutinas de atención de interrupciones

Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

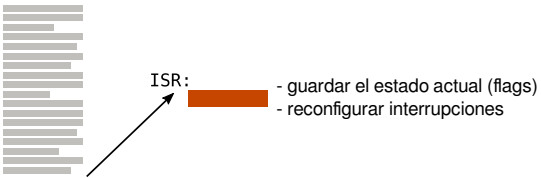


El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.

Rutinas de atención de interrupciones

Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:



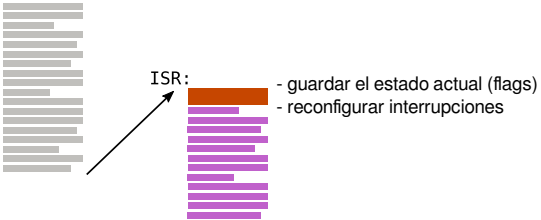
El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.

Rutinas de atención de interrupciones

Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.



Rutinas de atención de interrupciones

Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

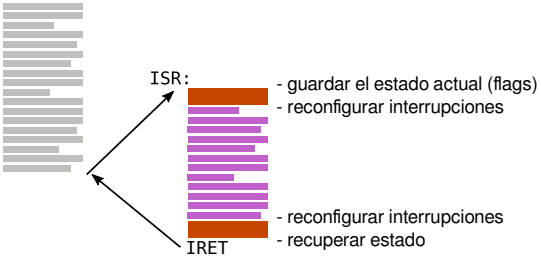
El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.

Rutinas de atención de interrupciones

Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.

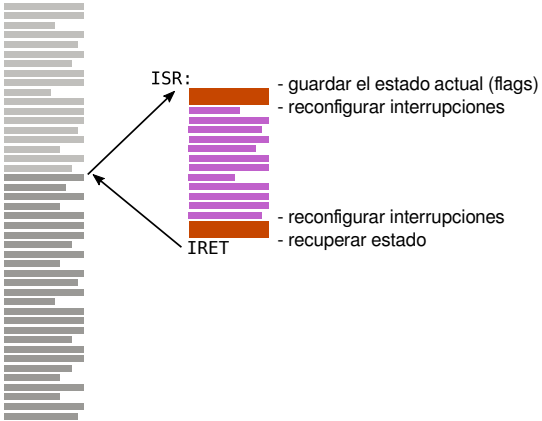


Rutinas de atención de interrupciones

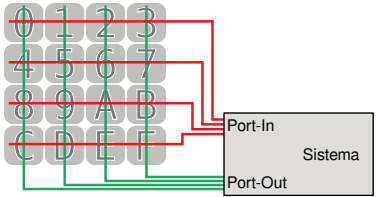
Las rutinas de atención de interrupciones son ejecutadas por el procesador cuando una interrupción es generada. El procesador guarda una **tabla** con la dirección donde comienza cada una de las rutinas de atención de interrupciones para cada interrupción posible.

Una rutina de atención de interrupciones tipo tiene la siguiente estructura:

El identificador de la interrupción se usa como **índice** para encontrar el código de la rutina a ejecutar.



Ejemplo: “Teclado númerico”



Se tienen dos puertos `portIn` y `portOut` conectados a un teclado. Ambos mapeados a memoria. Cualquier cambio en `portIn` genera una interrupción. Se desea agregar a un `buffer` las teclas presionadas.

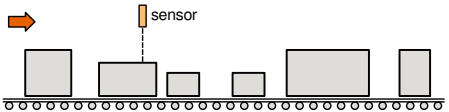
Rutina principal

```
main:
    while(true)
        MEM[portOut] = 0x1
        MEM[portOut] = 0x2
        MEM[portOut] = 0x4
        MEM[portOut] = 0x8
```

Rutina de atención de interrupciones

```
int:
    horizotal = MEM[portIn]
    vertical = MEM[portOut]
    if( horizotal == 1 ) value = 0
    if( horizotal == 2 ) value = 1
    if( horizotal == 4 ) value = 2
    if( horizotal == 8 ) value = 3
    if( vertical == 1 ) value += 0
    if( vertical == 2 ) value += 4
    if( vertical == 4 ) value += 8
    if( vertical == 8 ) value += 12
    addBuffer(value)
    iret
```

Ejemplo: “Cinta trasportadora”

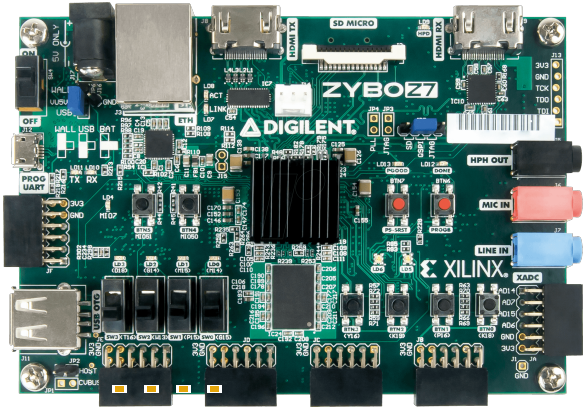


Se tiene un sensor que detecta la presencia de un objeto. Reporta `0xFF` si hay un objeto y `0` en caso contrario. El sensor se encuentra mapeado a memoria. Se desea contar la cantidad de objetos que son detectados por el sensor.

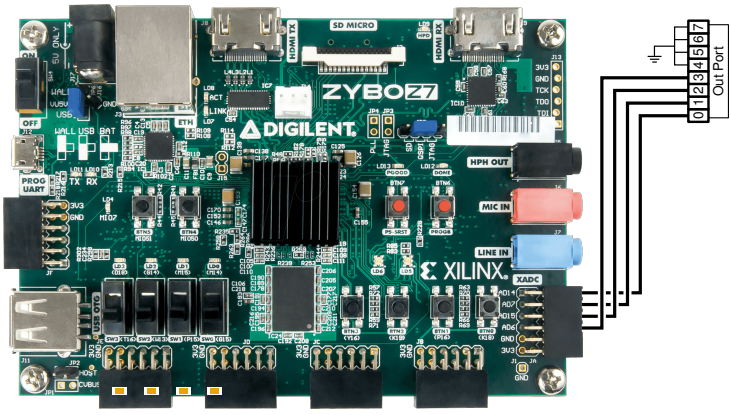
Rutina principal

```
contador = 0
anterior = 0
    main:
        while(true):
            if( anterior != sensor )
                if( sensor == 0xFF )
                    contador = contador + 1
                anterior = sensor
```

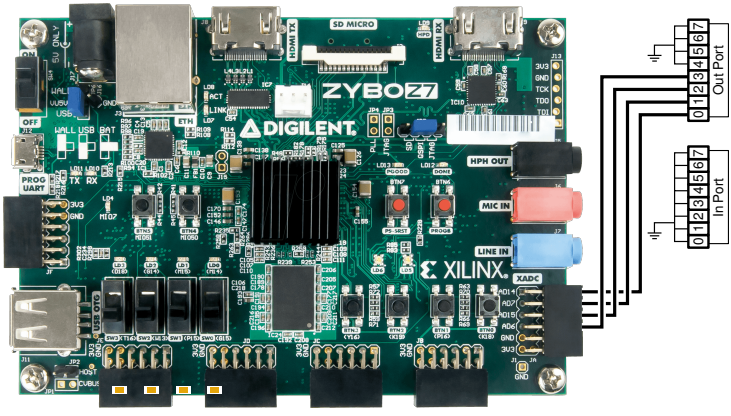
Ejemplo: OrgaSmallWithIO



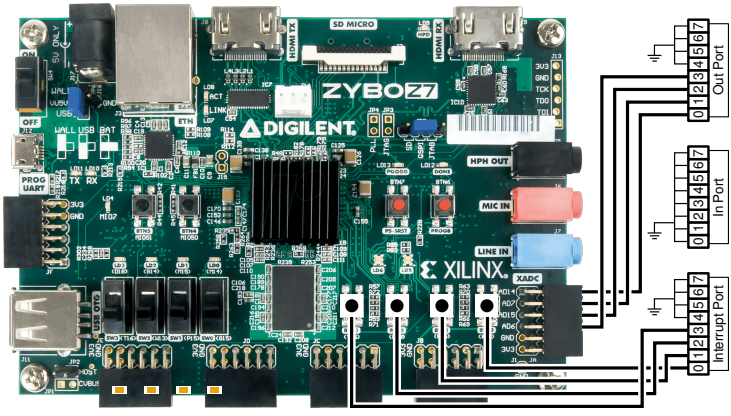
Ejemplo: OrgaSmallWithIO



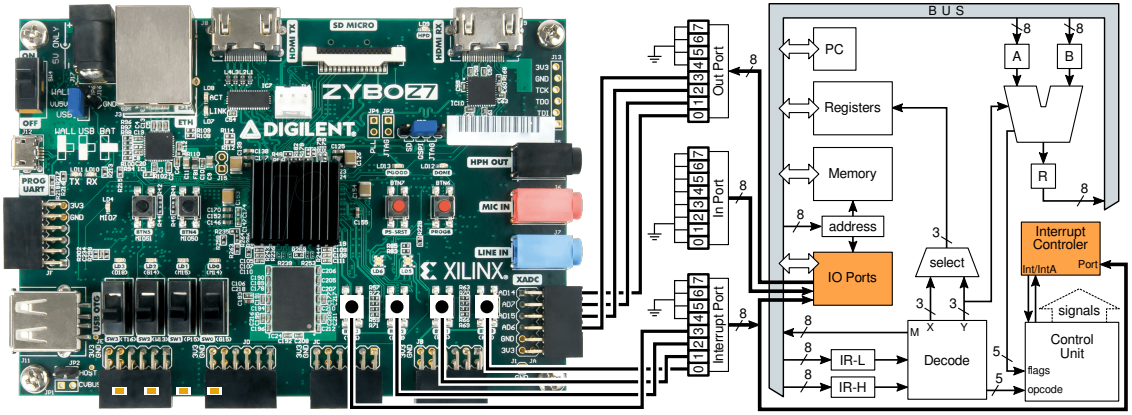
Ejemplo: OrgaSmallWithIO



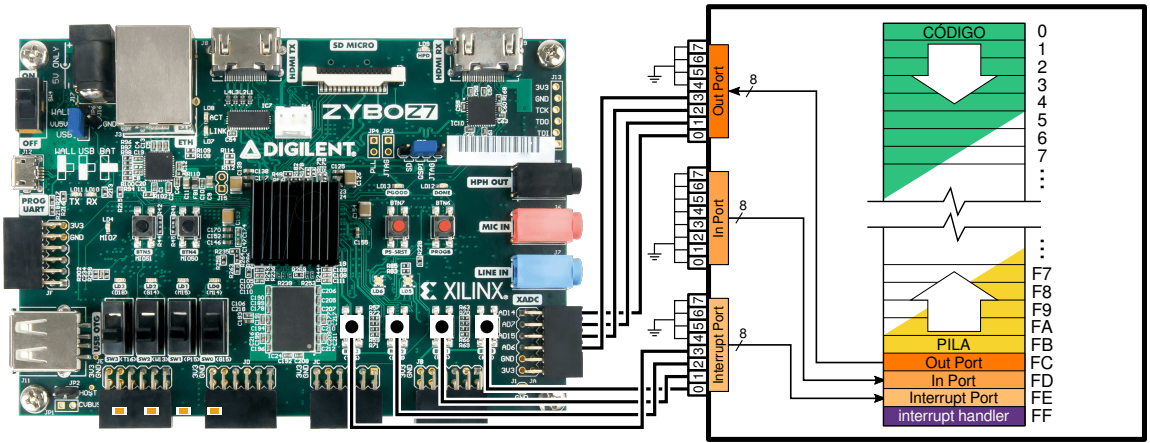
Ejemplo: OrgaSmallWithIO



Ejemplo: OrgaSmallWithIO



Ejemplo: OrgaSmallWithIO



Ejemplo: OrgaSmallWithIO

Rutina Principal

```
main:
    while(true):
        MEM[portOut] = 1
        sleep()
        MEM[portOut] = 0
        sleep()

sleep:
    v = MEM[data]
    while(v != 0):
        v = v-1
        MEM[portOut] = MEM[portOut] + 0x8
```

Rutina de atención de interrupciones

```
int:
    p = MEM[portInterrupt]
    if(p == 1):
        MEM[data] = MEM[data]+1
    if(p == 2):
        MEM[data] = MEM[data]-1
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue
```

```
sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue
```

```
sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue

data:
    DB 0x0F


sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue

data:
    DB 0x0F

sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

```
interrupt_handler:
    PUSH |R7|, R0
    PUSH |R7|, R1
    PUSH |R7|, R2
    PUSH |R7|, R3
    PUSH |R7|, R4
    SET R0, 0x1
    SET R1, 0x2
    LOAD R3, [0xFE]
    CMP R3, R0
    JZ inc
    CMP R3, R1
    JZ dec
    end_interrupt:
    POP  |R7|, R4
    POP  |R7|, R3
    POP  |R7|, R2
    POP  |R7|, R1
    POP  |R7|, R0
    RETI |R7|
    inc:
        LOAD R3, [data]
        ADD R3, R0
        STR [data], R3
        JMP end_interrupt
    dec:
        LOAD R3, [data]
        SUB R3, R0
        STR [data], R3
        JMP end_interrupt
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue

data:
    DB 0x0F

sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

```
interrupt_handler:
    PUSH |R7|, R0
    PUSH |R7|, R1
    PUSH |R7|, R2
    PUSH |R7|, R3
    PUSH |R7|, R4
    SET R0, 0x1
    SET R1, 0x2
    LOAD R3, [0xFE]
    CMP R3, R0
    JZ inc
    CMP R3, R1
    JZ dec
    end_interrupt:
    POP  |R7|, R4
    POP  |R7|, R3
    POP  |R7|, R2
    POP  |R7|, R1
    POP  |R7|, R0
    RETI |R7|
    inc:
        LOAD R3, [data]
        ADD R3, R0
        STR [data], R3
        JMP end_interrupt
    dec:
        LOAD R3, [data]
        SUB R3, R0
        STR [data], R3
        JMP end_interrupt
```

Ejemplo: OrgaSmallWithIO

```
; Memory
; 0x00 = entry point
; ...
; 0xFB = stack base
; 0xFC = port Output
; 0xFD = port Input
; 0xFE = port Interrupt
; 0xFF = Interrupt rutine pointer

; set STACK
SET R7, 0xFB

; Set Interrupt Rutine
SET R0, interrupt_handler
STR  [0xFF], R0

; Set Interrupt Flag
SET R0, 0x10
LOADF R0

; main
SET R0, 0x0
SET R1, 0x1
```

```
; loop
whileTrue:
    STR [0xFC], R0
    CALL |R7|, sleep
    STR [0xFC], R1
    CALL |R7|, sleep
JMP whileTrue

data:
    DB 0x0F

sleep:
    SET R3, 0x8
    LOAD R2, [data]
    ciclo:
        SUB R2, R1
        CMP R2, R0
        JZ end_sleep
        ; start SLEEP COUNTER
            LOAD R4, [0xFC]
            ADD R4, R3
            STR [0xFC], R4
        ; end SLEEP COUNTER
        JMP ciclo
    end_sleep:
    RET  |R7|
```

```
interrupt_handler:
    PUSH |R7|, R0
    PUSH |R7|, R1
    PUSH |R7|, R2
    PUSH |R7|, R3
    PUSH |R7|, R4
    SET R0, 0x1
    SET R1, 0x2
    LOAD R3, [0xFE]
    CMP R3, R0
    JZ inc
    CMP R3, R1
    JZ dec
    end_interrupt:
    POP  |R7|, R4
    POP  |R7|, R3
    POP  |R7|, R2
    POP  |R7|, R1
    POP  |R7|, R0
    RETI |R7|
    inc:
        LOAD R3, [data]
        ADD R3, R0
        STR [data], R3
        JMP end_interrupt
    dec:
        LOAD R3, [data]
        SUB R3, R0
        STR [data], R3
        JMP end_interrupt
```

Ejemplo: OrgaSmallWithIO

¡Manos a la obra!

Bibliografía

- Tanenbaum, “Organización de Computadoras. Un Enfoque Estructurado”, 4ta Edición, 2000.
  - **Seleccionados**
    - 2.4 Entrada/Salida - Páginas 89 - 92
    - 3.4 Chips de CPU y buses - Páginas 154 - 170
    - 3.7 Interfaces - Páginas 193 - 197
    - 5.5.7 Entrada/Salida - Páginas 356 - 359
    - 5.6.4 Trampas - Página 379
    - 5.6.5 Interrupciones - Páginas 379 - 383

- Null, “Essentials of Computer Organization and Architecture”, 5th Edition, 2018.
  - **Chapter 7 - Input/Output Systems**
    - 7.4 I/O Architectures
    - 7.4.1 I/O Control Methods
    - 7.4.3 I/O Bus Operation

Ejercicios

Con lo visto, ya pueden resolver todos los ejercicios de la guía de ejercicio número 5.

¡Gracias!

Recuerden leer los comentarios adjuntos
en cada clase por aclaraciones.