Tema 8: Mecanimos de Entrada/Salida

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2018 September 12		CA

Índice

1. Temario **1**

2. Bibliografia **1**

3. Periféricos **1**

3.1. Ejemplos . . . 1

3.2. Modelo . . . 1

4. Teclado **2**

5. Arquitectura Computadora **2**

5.1. Von Neumann . . . 2

5.2. Conexión CPU-E/S . . . 3

5.3. Controlador I/O . . . 3

5.3.1. Introducción . . . 3

5.3.2. Puertos . . . 4

5.4. Espacio de direcciones . . . 5

5.4.1. Memory-Mapped I/O (MMIO) . . . 5

5.4.2. Port mapped I/O (PMIO) . . . 5

5.4.3. Direcciones de los periféricos . . . 5

5.5. Buses . . . 6

5.6. Analisis: Portatil Lenovo - Disco Duro . . . 7

6. Programa E/S **8**

6.1. Módulo fuente . . . 8

6.1.1. ISA . . . 8

7. Driver: Sistema Operativo **8**

7.1. Gestor E/S: jerarquía . . . 8

7.2. Código Fuente . . . 8

7.3. Concepto . . . 9

7.4. Utilizacion del Driver . . . 9

8. Mecanismos de Implementación de la Interfaz E/S **9**

8.1. Introduccion . . . 9

8.2. Sincronización por Encuesta . . . 10

8.3. Sincronización por Interrupción . . . 10

8.4. Direct Memory Access (DMA) . . . 11

8.5. Channel I/O . . . 12

8.5.1. Memory Shared . . . 12

8.5.2. Memory Independent . . . 12

9. Sincronizacion por Interrupcion **13**
9.1. Concepto . . . 13
9.2. Mecanismo de Interrupcion . . . 13
9.3. Controlador de Interrupciones . . . 13
9.3.1. PIC . . . 13
9.3.2. NMI . . . 15
9.3.3. Intel . . . 15
9.4. Gestor de Interrupciones . . . 15
9.5. Tipos de Interrupciones . . . 16
9.6. Tabla de los Vectores de interrupciones . . . 16
9.6.1. Modo Real: Tabla IVT . . . 16
9.6.2. Modo Protegido: Tabla IDT . . . 18
9.6.3. IRQ . . . 19
9.6.4. Linux . . . 20
9.6.5. Lineas compartidas . . . 20

10. Acceso Directo a Memoria DMA **20**
10.1. Funcionalidad . . . 20
10.2. Transferencias . . . 21
10.3. Sincronización . . . 21
10.4. Operación del controlador DMA . . . 21
10.4.1. Secuencia de pasos a nivel alto . . . 21
10.4.2. Secuencia de pasos a nivel bajo . . . 21
10.5. Problemas de coherencia en la memoria cache . . . 22

11. Buses **22**
11.1. ISA . . . 22
11.2. PCI . . . 22
11.3. North-South Bridge . . . 22
11.4. Chipset x58 . . . 23

12. Programacion de rutinas de entrada/salida **23**
12.1. Software jerarquico del sistema operativo . . . 23
12.2. Instruction Set Architecture . . . 23
12.2.1. Intel Manual . . . 24
12.3. Programación del Controlador de Interrupciones Programable . . . 24
12.4. Driver del Teclado . . . 25
12.5. paralell port . . . 25
12.5.1. Desde Espacio de Usuario . . . 25
12.6. Serial communication RS-232 . . . 26

13. Ejercicios **27**

1. Temario

1. Sistema de entrada / salida
    a. Sincronización por encuesta
    b. Sincornización por interrupción
    c. Vector de interrupciones
    d. Acceso directo a memoria DMA
    e. Programación en lenguaje ensamblador de rutinas de entrada/salida

2. Bibliografia

- Libro de Texto:
    - Estructura y Organización de Computadores. William Stalling: Capítulo 7

3. Periféricos

3.1. Ejemplos

- Teclado
- Monitor
- Disco Duro
- Red
    - LAN
    - Wifi
- Periférico externo
    - Pen-Drive
- De cada ejemplo info de:
    - modelo y link a características
    - interfaz: bus eléctrico, protocolo de comunicaciones
    - ancho de banda

3.2. Modelo

- Media : Magnético (HD), Mecánico (Robot), Optico (CD), Eléctrico (pen drive), etc
    - Electrónica analógica.
- Driver HW
    - Interfaz con el media:
        - las señales que actuan sobre el media son de distinto tipo: optica (luz), mecánica (pneumático), acústica, etc. Estas señales se obtienen normalmente de la transformación de una señal eléctrica: interfaz eléctrico/optico, eléctrico/mecánico, eléctrico/acústico

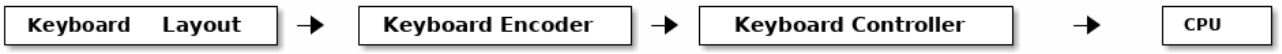
  - Ej: Un altavoz
  - Ej: Un Láser con el disco óptico.
  - Ej: El modulador electrónico del láser

- Controlador del Periférico (**MCU: MicroController Unit**)
  - Imagen de un controlador de disco
  - El Controlador da órdenes al Driver HW
  - Es un secuenciador que interpreta **Comandos** (lenguaje específico para tareas del periférico) cuya ejecución realizará funciones propias del periférico.
    - Lenguaje de comandos. Command Set Architecture (CSA) ¿ISA?
      - Lenguaje SCSI
      - Lenguaje ATA / ATAPI
      - ATA Command Set (ACS): ejemplo de comandos IDENTIFY, READ DMA, WRITE DMA and FLUSH CACHE commands
    - comandos de transferencia de datos, de control (operaciones mecánicas como girar), de test (estado del periférico: conectado, desconectado)
  - Ej: en el caso de un disco el comando "girar a determinadas revoluciones". El disco integra un secuenciador propio, un MCU.

- Firmware
  - El set de comandos del periférico son interpretados por el software (firmware) cargado en la memoria del controlador MCU. Dicho software ha sido grabado por el fabricante del periférico. El usuario únicamente podrá escribir en el periférico algunos parámetros de configuración del periférico siendo accesible el Firmaware unicamente por el fabricante.

4. Teclado

- Estructura



- Códigos

- Driver

5. Arquitectura Computadora

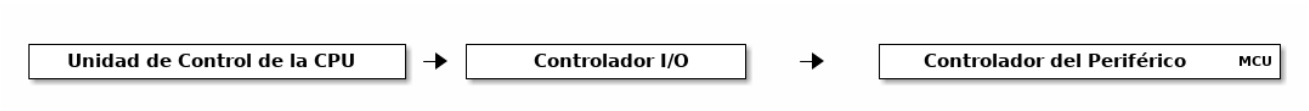
5.1. Von Neumann

- 3 unidades básicas
- Controlador i/o

  - Controlador auxiliar, no central, con dedicación específica a las operaciones E/S.
- Subsistema de Entrada/Salida : es uno de los 3 componentes del modelo Von Neumann
- Es necesario acceder a la máquina para:
  - Introducir el programa : Desde un soporte de almacenamiento (papel,disco,etc) ha de cargarse el programa en la memoria
  - Extraer el resultado generado por la máquina: Desde la memoria los resultados han de almacenarse en un soporte de almacenamiento (disco, impresora, etc), visualizarse (pantalla, etc), transferirse (red, etc).
- Dispositivos Periféricos
  - Son recursos hardware que complementan y extienden los servicios del tándem CPU-MEMORIA facilitando las tareas del progamador y del usuario.
  - Gran diversidad: teclado, monitor, ratón, discos, tarjeta video, tarjeta red, ...
  - diferencia de complejidad entre un teclado y un disco duro
- La CPU normalmente es un recurso único compartido por todos los programas y por todos los periféricos.

5.2. Conexión CPU-E/S

- La arquitectura está formada, por lo tanto, por dos controladores: CPU y MCU. La CPU tiene un controlador generalista (CPU) mientras que el periférico tiene un controlador (MCU) muy específico. El lenguaje máquina de la CPU es generalista mientras que el lenguaje máquina del periférico es muy específico.
  - El periférico se comporta como una máquina **servidor** con su propio procesador. Podemos hablar de la máquina **host** (anfitrión) y de la máquina **server** . El controlador host es la CPU de la computadora y el controlador server es la MCU del periférico.
  - La CPU no se comunica directamente con el MCU sino que delega la tarea de los periféricos a procesadores no centrales, es decir, a los controladores I/O. La arquitectura típica de la computadora es la de un Procesador Central y un Set de controladores i/o que Intel denomina **Chipset**



  - Ejemplo: Disco Duro
    - Seagate Momentus 7200.4 500GB 7.2K 2.5-inch SATA Hard Drive ST9500420AS
    - video del movimiento del brazo
    - Disk buffer: memoria interfaz entre la transferencia del drive y la transferencia i/o del puerto.
    - Controlador Atmel casero: Disco Sata con controladora Atmel e interfaz ethernet.

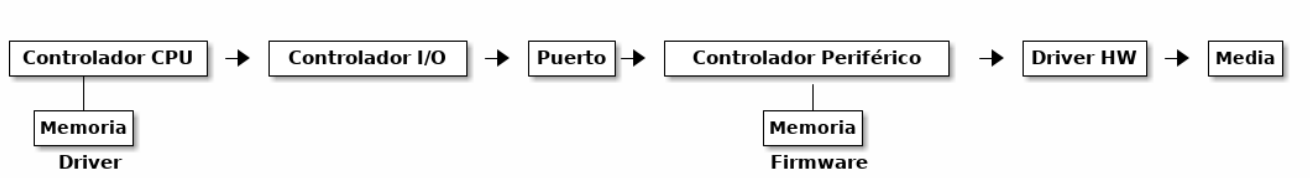
5.3. Controlador I/O

5.3.1. Introducción

- Periférico remoto:
  - Ej:PC---→SATA---→Compact Disc
- Controlador I/O del PC :
  - NO es la Unidad de Control de la CPU
  - Es uno de los 3 elementos básicos de la arquitectura Von Neumann
  - La CPU delega en otro controlador denominado controlador I/O la ejecución de las instrucciones máquina de entrada/salida.

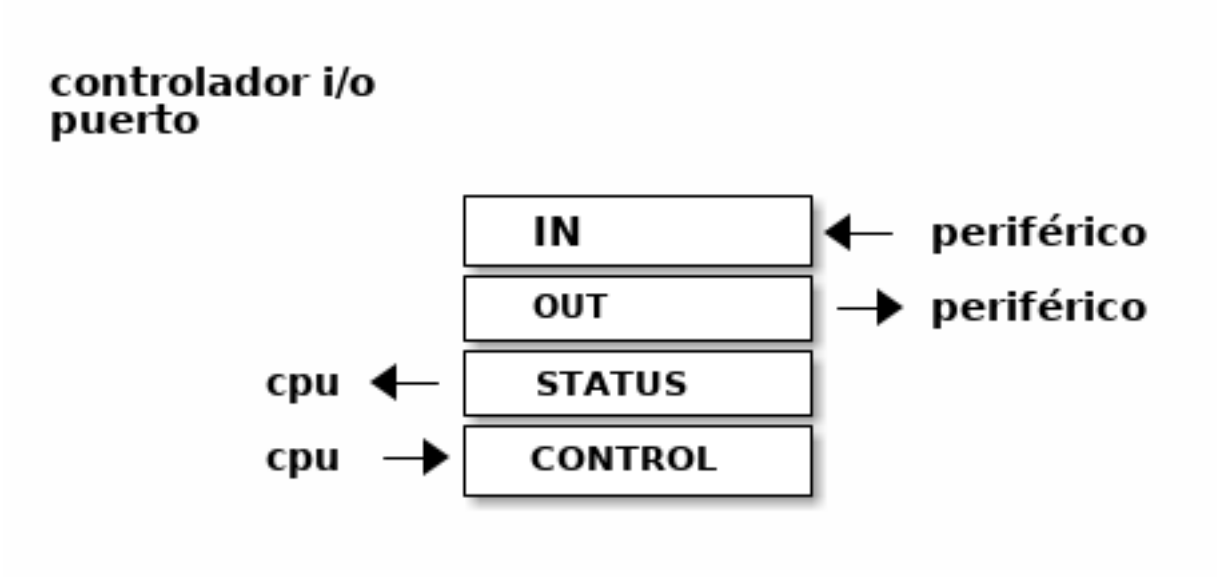
- Es necesario una INTERFAZ entre la CPU y el PERIFERICO
- ISA:Instrucciones máquina de entrada salida de la cpu: de lectura (IN) y de escritura (OUT). Comunicación en ambos sentidos.
  - Se transfieren tanto los DATOS como los COMANDOS del periférico.
- William denomina al controlador I/O con el nombre Módulo de E/S
- Es el controlador I/O el que transfiere los comandos y los datos al controlador del periférico.
  - El controlador del periférico interpretará los COMANDOS recibidos del controlador I/O y escribirá o/y leerá los DATOS.
  - Ejemplo:The Advanced Host Controller Interface (AHCI) is a technical standard defined by Intel that specifies the operation of Serial ATA (SATA) io controller (host bus adapters).

- Estructura



5.3.2. Puertos

- Los puertos son registros de memoria implementados en el controlador i/o.
  - Un puerto está formado por distintos tipos de registros: entrada de datos, salida de datos, estado del periférico, control del periférico



- El **controlador I/O** controla y ejecuta las comunicaciones a través de sus puertos.
  - Ej: Controlador I/O con puerto SATA
- **Puerto** de comunicaciones: Acceso al otro interlocutor (el periférico en este caso)
- Un controlador I/O puede tener varios puertos y controlar las comunicaciones con varios periféricos.
- Un puerto puede ser compartido por varios periféricos
- Linux
  - `cat /proc/ioports`

5.4. Espacio de direcciones

- Las direcciones i/o del puerto del controlador i/o se puede implementar de dos formas:
  - puertos mapeados en la memoria principal
  - direcciones de los puertos en un espacio diferente de la memoria principal: espacio i/o.

5.4.1. Memory-Mapped I/O (MMIO)

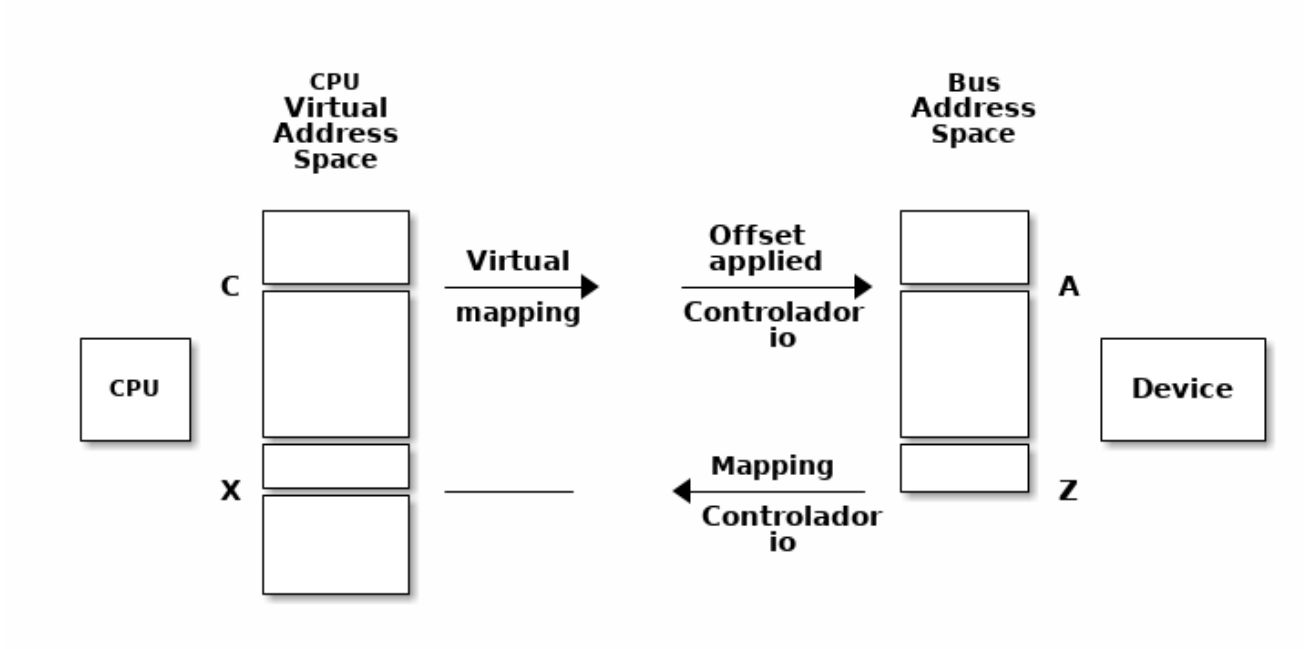
- Main Memory Address Space
- El bus del sistema shares memory address space between I/O devices and program memory
- Interface to the I/O is treated as a set of primary memory locations (Memoria principal)
- Software drivers determine meaning of data stored or retrieved.
- Loss of some memory space (8086 - 300K) because it is reserved for I/O interfaces. Less important with 4GiB address space.
- All instruction modes available → Todo el repertorio de instrucciones, no solo IN,OUT.
- May slow overall memory bus access down.
- Can limit or complicate contiguous memory range.
- Original x86 architecture had a 1MiB boundary because I/O was mapped above 640K.

5.4.2. Port mapped I/O (PMIO)

- I/O Address Space
- CPU has separate set of instructions that access specific pin on CPU that act as ports or that cause a demux to connect the address and data pin-outs to a different set of lines tied to i/o.
- Separate bus tied to I/O devices. CPU can go back to using memory bus while I/O devices responds.
- Adds complexity to CPU design.
- Often limited set of instructions. May need to write to memory before other actions can be taken.

5.4.3. Direcciones de los periféricos

- Driver
  - El programa driver es un proceso i/o que utiliza el mecanismo de memoria virtual igual que el resto de procesos.
  - Mapa en el fichero */proc/iomem*
- Periférico
- Al espacio de direcciones utilizado por los periféricos se le denomina **bus address**
- Hará falta mapear, traducir, direcciones bus a direcciones virtuales de la cpu
  - Mapa en el fichero */proc/ioports*



5.5. Buses

- Tipos de buses
  - del procesador: bus interno a a la cpu
  - de memoria: bus entre el controlador de memoria y la memoria principal
  - del sistema: bus externo a la cpu para el interconexiado de dispositivos externos como la memoria principal y los controladores i/o de los periféricos.
  - local: bus i/o corto que permite elevados anchos de banda
  - de expansión: bus i/o largo que permite la conexión de múltiples *tarjetas*
  - periférico: bus i/o que permite conectar dispositivos externos a la computadora

- Bus i/o local; vlb,PCI, AGP
- Bus i/o de expansión:ISA, EISA
  - Conexión directa de la tarjeta i/o al bus de expansión de la placa base a través de *slots*:
- Bus periférico: SCSI, SATA, USB, RS232
  - Conexión externa a través de un *cableado*
- La arquitectura del bus io la componen
  - Interfaz (cable y conector)
  - Protocolo de comunicaciones: set of standardized rules for consistent interaction between system and i/o devices, including physical properties, access methods, data formats, etc. El Bus da nombre al protocolo.
  - Lenguaje de comandos
- Ejemplos prácticos

- ISA
  - Industry Standard Architecture
  - PC/XT 8086 (1983) 8 bits
    - 4 canales DMA
  - PC/AT i286 (1984) 16 bits
    - 16 MB/s
    - 7 canales DMA
    - 11 líneas IRQ
- EISA
  - Extended Industry Standard Architecture
  - PC Clon: i386-i486 (1988)
  - 32 Bits
  - Alternativa de los clónicos al propietario MCA de IBM en su PS/2
  - 33 MB/s de velocidad de transferencia para buses maestros y dispositivos DMA
  - 7 canales DMA
  - 15 líneas IRQ
- MCA
  - Micro Channel Architecture
  - IBM PS/2 (1987)
  - 32 bits
- PCI: Peripheral Component Interconnect
- PCI Express
- listado de anchos de banda

- http://www.karbosguide.com/hardware/module2b2.htm
  - El controlador i/o se conecta indirectamente al bus del sistema (CPU-MP) a través de los puentes (bridges)
- Intel
  - Intel ha evolucionado de los puentes ICH con el puente Sur y Norte a un Concentrador Central PCH
    - Observar que la CPU integra el controlador de memoria integrado (IMC) y controladores i/o de video (PCI-E Graphics)

5.6. Analisis: Portatil Lenovo - Disco Duro

- Ruta de la transferencia de datos entre el disco duro y la memoria principal en la computadora Lenovo T520
- Disco (ATA disk, ST9500420AS Seagate)→ Driver Mecánico/Electrico/Magnético → Micro del Disco (SATA Interface, Seagate) → Bus i/o serie(SATA 6Gb/s)→ Host Adapter (Platform_Controller_Hub PCH, ChipSet 200C/6 Series, SATA AHCI Controller)→ Flexible Display Interface (FDI) → CPU (Intel Core i5)
- SATA: Serial Advanced Technology Attachment is a computer bus interface that connects host bus adapters (controladora de disco) to mass storage devices (MCU, MicroControllerUnit) such as hard disk

Lenovo T520

Figura 1: Lenovo T520

6. Programa E/S

- Programmed i/o (PIO)
  - Las transferencias de datos mediante mecanismos de E/S por consulta la realiza un programa i/o (PIO) que ejecuta la CPU. La CPU en cada transferencia de datos entre la memoria y el periférico debe de ESPERAR a que dicha transferencia termine.

6.1. Módulo fuente

- Transferencia de 512 bytes entre el puerto 0x380 y un buffer.

```
      mov %bx,buf ; destination address. BX es un puntero a un buffer
      mov $512,%si ; count. Número de bytes a transferir
      mov $0x380,%dx ; source port. DX es un puntero al puerto
loop:
      in %dx,%al ; get byte from i/o port. AL<-DX
      mov %al,(%bx) ; store in buffer      M[bx]<-AL
      inc %bx ; next memory location in buf
      dec %si ; decrement bytes left
      jnz loop
```

6.1.1. ISA

- IN: leer un dato del puerto
- OUT: escribir un dato en el puerto

7. Driver: Sistema Operativo

7.1. Gestor E/S: jerarquía

- La gestión de las operaciones E/S las realiza el Sistema Operativo
- La estructura del programa gestor E/S del sistema operativo se basa en una estructura jerárquica por niveles:
  - Nivel más bajo: controlador sw (módulo driver) del controlador hw i/o del periférico.
  - Nivel más alto: Sistema virtual de ficheros. Las aplicaciones acceden a los periféricos mediante la abstracción de estos en ficheros virtuales.

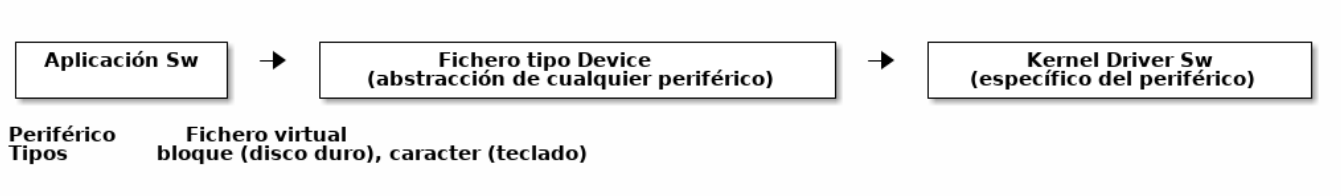
7.2. Código Fuente

- Pseudo-código

```
While (STATUS == BUSY)
   ; // wait until device is not busy . Puerto ocupado.
Write data to DATA register // dato a transmitir el puerto out
Write command to COMMAND register  // registro de control
   // Doing so starts the device and executes the command
While (STATUS == BUSY)
   ; // wait until device is done with your request
```

7.3. Concepto

- El programa que implementa las funciones del periférico es un *módulo del kernel* denominado DRIVER
  - Driver del teclado, monitor, disco duro,
- Capas SW:



- Ejemplo: Escritura de un fichero en el disco duro
  - write → syscall → OUT → comando propio del HD

  - Espacio de usuario: write (función de escritura de datos), syscall (llamada al módulo gestor de E/S del Sistema Operativo)
  - Espacio kernel del S.O.: Driver: Orden interpretada por la CPU y ejecutada por el controlador I/O para transferir datos (comandos y datos) entre Memoria y el Controlador Periférico
  - Espacio periférico: Comandos a Interpretar por el Periférico (Firmware) y transferencia de Datos.

7.4. Utilizacion del Driver

- El driver está protegido por el Sistema Operativo. Hay funciones como ioctl que permite al usuario interactuar con el driver.
  - La interfaz entre el usuario y el driver son las llamadas al sistema operativo.
    - Mediante la instrucción máquina SYSCALL (x86-64) o int 0x80 (x86-32) llamamos indirectamente a las funciones del driver a través del sistema operativo.
- Ejemplo
  - Imprimir en la pantalla: open, write, close → open y close interactuan con el sistema de ficheros virtual.

8. Mecanismos de Implementación de la Interfaz E/S

8.1. Introduccion

- All data manipulation not directly performed in the CPU or between CPU and primary memory is I/O.
- PIO:Polling
- Interruption
- DMA: Direct Memory Access

8.2. Sincronización por Encuesta

- Polling: encuesta
- Query: encuesta
- Mecanismo
  - Comprobación del estado o *encuesta-polling*
    - La CPU consulta el registro de estado de cada puerto al que están conectados los periféricos. Comprueba si algún periférico requiere el servicio de la CPU. Reserves a register for each I/O device. Each register is continually polled to detect data arrival.
  - Es necesario ejecutar programas de atención al periférico cuando este lo requiera: sincronización
  - El anfitrión consulta el bit de estado del controlador i/o
  - Identificación
    - Una vez aceptada la petición del cliente (controlador i/o)
      - ◇ El controlador identifica el periférico que solicita el servicio
      - ◇ Comunica al S.O. qué periferico
- Estructura
  - CPU:
    - Ejecuta el programa i/o: un programa que controla DIRECTAMENTE la operación E/S: Programmed I/O → PIO
    - realiza las transferencias entre la memoria principal y el controlador i/o
    - *espera* al periférico hasta que termine. La CPU espera hasta que concluya la operación E/S.
  - Memoria principal: almacena el programa i/o
  - Controlador i/o
    - Puerto: Un puerto está compuesto por REGISTROS del tipo datos, control, test
    - transfiere los datos al periférico

8.3. Sincronización por Interrupción

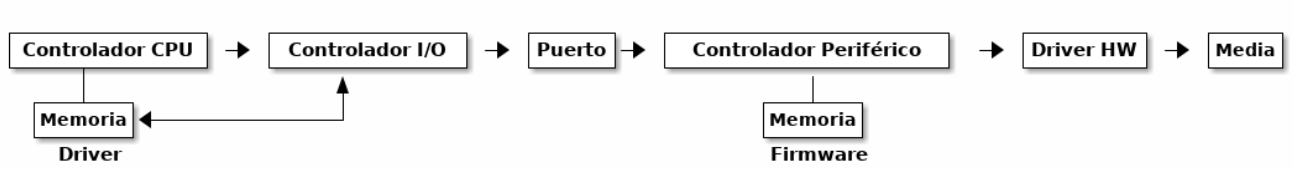
- Interrupt-Driven I/O (Mecanismo de E/S por Interrupción)
- Estructura
  - CPU:
    - Ejecuta el programa de atención a la interrupción. Un programa que controla DIRECTAMENTE la operación E/S.
    - realiza las transferencias entre la memoria principal y el controlador i/o
    - **no espera** al periférico hasta que termine. Es *interrumpido* cada vez que es necesario realizar una transferencia
      - ◇ Al finalizar el ciclo de instrucción de cada instrucción que ejecuta la CPU, se comprueba si la señal de petición de interrupción está activada.
  - Memoria principal: almacena el programa i/o
  - Controlador i/o
    - Puertos: datos, control, test
    - transfiere los datos al periférico
- allows the CPU to do other things until I/O is requested
  - Interrupt request - Driven I/O (Still PIO - CPU has to move data)
  - I/O devices can request the attention of CPU with an interrupt at any time, but only when needed.
  - CPU can dedicate extended time for particular device.

- CPU does not have to check in on I/O that does not need attention.
- CPU can delay processing of I/O request.
- Newer systems - cpu hands off transfer of data to secondary controller, which only interrupts cpu on completion of task or problem.
- Requires external circuitry.
  - e.g 8259A programmable interrupt controller (PIC). CPU may have to communicate with the PIC to identify requesting device.

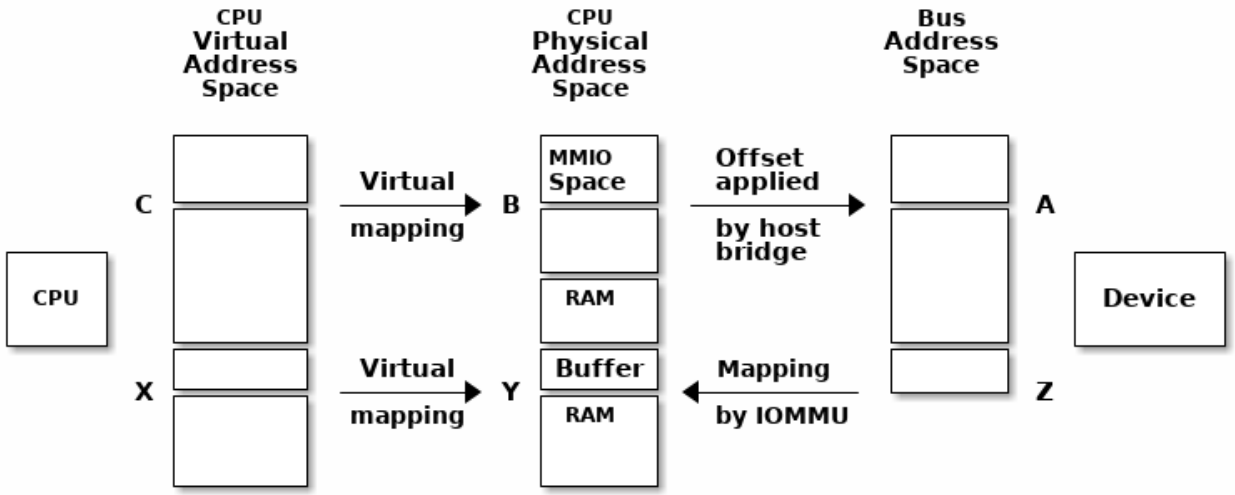
- Programmed input/output (PIO) is a method of transferring data between the CPU and a peripheral such as a network adapter or an ATA storage device. In general, programmed I/O happens when software running on the CPU uses i/o instructions that access I/O address space to perform data transfers to or from an I/O device. This is in contrast to Direct Memory Access (DMA) transfers.
- The best known example of a PC device that uses programmed I/O is the ATA interface;

8.4. Direct Memory Access (DMA)

- Estructura



- Mapeo de direcciones
- Hará falta una unidad hardware de traducción de direcciones bus a direcciones físicas : iommu



- Estructura
  - CPU:
    - Ejecuta el programa i/o. El programa no controla la transferencia pero sí la inicializa (número de bytes a transferir, localización en la memoria principal, localización en el periférico, control errores, etc)

- Cede el control de las transferencias al controlador DMA (DMAC)., offloads I/O processing to a special-purpose chip that takes care of the details. La transferencia la controla y realiza el DMAC por Hardware → No es por programa como el PIO.

- Memoria principal: almacena el programa i/o
- Controlador i/o
  - es el controlador DMA
  - Puertos: los puertos ahora no son para los datos de transferencia, unicamente para el control CPU-DMA
  - transfiere los datos entre la memoria principal y el periférico
  - el controlador no espera al periférico
  - Direct Memory Access controller.

```
Handles I/O interaction without the intervention of the CPU after ↩
    initial
  CPU interaction. Uses interrupts to report status back to CPU.
Requires separate arbitration protocol - shares buses with CPU.
Predefined standardized tasks.
CPU NOT occupied but may have to compete for resources.
```

8.5. Channel I/O

- uses dedicated I/O processors
  - Channel I/O (Mainframe or Supercomputer)
    - Estructura: integra la unidad DMA más un procesador específico.
    - Programable: ejecuta el *channel program* almacenado en la memoria principal. (Diferencia con DMA).
    - Transfiere datos (Memoria principal <→ Periférico) independientemente de la CPU

8.5.1. Memory Shared

- Estructura
  - CPU
    - cede el control de las transferencias al procesador o canal i/o
  - Memoria principal: almacena el programa i/o
  - Procesador i/o
    - Es el canal i/o
    - Ejecuta el programa i/o almacenado en la memoria principal
    - Puertos: los puertos ahora no son para los datos de transferencia, únicamente para la control CPU-DMA
    - transfiere los datos entre la memoria principal y el periférico
  - Memoria Principal
    - Compartida entre la CPU y el Canal_IO

8.5.2. Memory Independent

- Estructura
  - CPU
    - cede el control de las transferencias al procesador o canal i/o
  - Memoria principal: almacena el programa i/o

- Procesador i/o
  - Es el canal i/o
  - Ejecuta el programa i/o almacenado en la memoria principal
  - Puertos: los puertos ahora no son para los datos de transferencia, unicamente para el control CPU-DMA
  - transfiere los datos entre la memoria principal y el periférico
- Memoria Principal
  - Accesible sólo por la CPU
- Memoria IO
  - Accesible sólo por el canal_IO

9. Sincronizacion por Interrupcion

- Extensión del apartado anterior sobre implementación de la interfaZ i/o driven-interruption.

9.1. Concepto

- El inconveniente del Polling es que la CPU realiza la consulta aunque el periférico no requiera sus servicios.
- El periférico toma la iniciativa y solicita la INTERRUPCION del programa que este ejecutando para ejecutar el programa requerido por el periférico

9.2. Mecanismo de Interrupcion

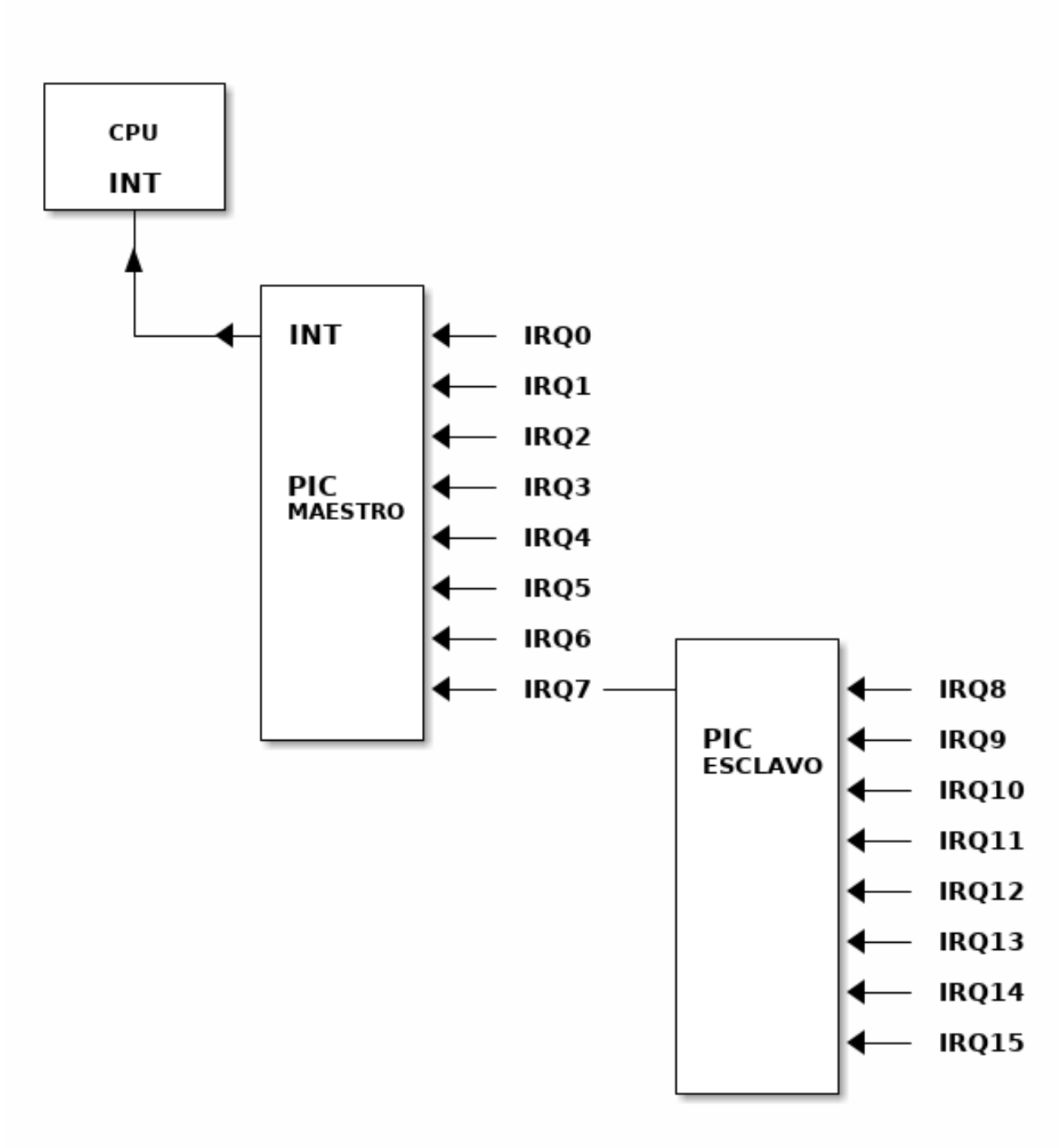
- El periférico a través de una línea eléctrica de entrada de la CPU, solicita al controlador i/o los servicios del kernel
  - El kernel va a ser **INTERRUMPIDO**
- La CPU tiene dos líneas de interrupción:
  - Línea Interrupt ReQuest *(IRQ)* : Maskable
  - Línea Non Maskable Interrupt *(NMI)*
  - La CPU en el ciclo de instrucción tiene en su última fase la fase de Chequeo de Interrupción
    - Si se está solicitando un servicio al kernel, la CPU entra en modo atención a la interrupción y pasa el control al módulo de **Gestión de Interrupciones** del Kernel
- Las lineas de los periféricos para solicitar la interrupción se denominan *(IRQ)*.

9.3. Controlador de Interrupciones

9.3.1. PIC

- Controlador de Interrupciones Programable
  - PIC : Programmable Interrupt Controller
    - Se utiliza en arquitecturas cuya CPU tiene un único núcleo.
    - Tiene como entrada todas las líneas de interrupción de los periféricos
    - Salidas: INT (petición de interrupción) y D0-D7 (Control,Status and Interrupt-Vector)
- Ejemplo : PIC 8259

- Conexión daisy-chain
  - Maestro - Esclavo



- Acciones que realiza el controlador PIC
  - Chequea si se activa alguna señal (Monitorización). En caso de activarse más de una se dar prioridad a la de menor nivel y procede:
    - **Convierte** la línea activada IRQn en un vector (0x00-0xFF)
    - Escribe en el puerto i/o del PIC el vector. El puerto es accesible por parte de la CPU. El vector se apunta a la entrada de una tabla que contiene un puntero a la rutina de atención a la interrupción (ISR)

    - Activa la señal INTR de la CPU
    - Si la CPU lee el valor del vector se desactiva la señal INTR

- A la línea IRQn le corresponde, por defecto según Intel, el vector *n+32*. Este mapeo se puede alterar programando el PIC.
  - A la línea IRQ0 le corresponde el vector 0x20
- Cada línea IRQ se puede desautorizar por programa pero dicha interrupción no se pierde.
- Mediante la instruccion `cli` se hace un clear del flag IF del registro EFLAGS, ignorando la CPU todas las interrupciones hardware.
  - Mediante la instrucción `sli` hacemos un set de IF.

9.3.2. NMI

- Es una entrada de la cpu.
- The NMI (is a hardware driven interrupt much like the PIC interrupts but the NMI goes diretcly the cpu, and not via the PIC controller.
- Aplicaciones
  - Temporizador watchdog
    - Es un temporizador que hay que poner a cero regularmente. Si la cpu está bloqueada no podrá resetear el temporizador y este generará un interrupción nmi con lo que el contador de programa se cargará con una dirección que apunta a la rutina de atención a la interrupción NMI cuya ejecución desbloqueará el estado de la cpu.

9.3.3. Intel

- Intel
  - 1º generation of interrupts (XT-PIC): only supported 15 interrupts.
  - 2º generation (IO-APIC): number of supported interrupts to 24.
    - APIC : Advanced Programmable Interrupt Controller: Utilizado en modernas arquitecturas multinúcleo.
  - 3º generation, MSI: number of available interrupts to 224.

9.4. Gestor de Interrupciones

- El Gestor de Interrupciones está implementado por el S.O.: entry.S en la versión linux 2.x
- Identifica al solicitador de la interrupción para poder ejecutar la rutina específica de antención a dicha interrupción.
- Anula la posibilidad de ser interrumpido por otros dispositivos no prioritarios, a través del flag IF , bit 9 del registro de control rflags de la CPU.
- Consulta el Vector de interrupciones (Tabla de punteros a las rutinas de atención a la interrupción.
- Salva el entorno del programa en ejecución que va a ser interrumpido.
- Activa la rutina Interrupt Service Routine *(ISR)*.
  - Autoriza nuevamente las interrupciones
  - Dicha rutina estará implementada en el módulo driver del kernel.

9.5. Tipos de Interrupciones

- Intel define dos tipos de Interrupt Signals
  - **Sincronas**
    - Son originadas por la propia CPU al final del ciclo de instrucción en la fase de interrupción
    - Se denominan *exceptions*
      - *Interrupt Software* : originadas por la instrucción `syscall`: Llamadas al Sistema
      - Originadas por un *Error*: Fault, Trap, Abort.
  - **Asíncronas**
    - Originadas por los periféricos o hardware que no es la CPU
    - Se denominan *interruptions* o *hardware interruptions*
      - Maskable: IRQ
      - Non-Maskable: NMI

9.6. Tabla de los Vectores de interrupciones

9.6.1. Modo Real: Tabla IVT

- En plataformas con S.O. al encender la computadora (arranque con bootloader) la CPU está operando inicialmente en *Modo Real* y en plataformas sin S.O. (arranque con BIOS) la cpu opera permanentemente en Modo Real. En plataformas con S.O. el arranque se inicia en modo real y se configura la computadora para pasar al modo protegido antes de cargar el S.O. en la memoria principal.
  - Real Mode :
  - Is a simplistic 16-bit mode that is present on all x86 processors: Equivale a comportarse como la antigua cpu 8086.
  - la cpu 8086 tiene 20 bits de direcciones y 16 bits de datos.
  - A real mode pointer is defined as a 16-bit segment address and a 16-bit offset into that segment
    - El segmento se expande a 20 bits multiplicando x4.
  - 2^{20}:El Código tiene que estar en el primer Mega de la memoria RAM
- Permite el acceso a funciones de la BIOS.
  - Tabla de interrupciones BIOS
  - Tecnología del PC

```
MOV AH, 0Eh   ; Imprime carácter en la pantalla
MOV AL, '!'   ; carácter a imprimir
INT 10h       ; Llamada a las funciones de video del BIOS
```

- Tabla IVT.

```
Interrupt Address       Type Description

00h     0000:0000h      Processor       Divide by zero
01h     0000:0004h      Processor       Single step
02h     0000:0008h      Processor       Non maskable interrupt (NMI)
03h     0000:000Ch      Processor       Breakpoint
04h     0000:0010h      Processor       Arithmetic overflow
05h     0000:0014h      Software        Print screen
06h     0000:0018h      Processor       Invalid op code
07h     0000:001Ch      Processor       Coprocessor not available
08h     0000:0020h      Hardware        System timer service routine
09h     0000:0024h      Hardware        Keyboard device service routine
```

```
0Ah     0000:0028h      Hardware        Cascade from 2nd programmable interrupt controller
0Bh     0000:002Ch      Hardware        Serial port service - COM post 2
0Ch     0000:0030h      Hardware        Serial port service - COM port 1
0Dh     0000:0034h      Hardware        Parallel printer service - LPT 2
0Eh     0000:0038h      Hardware        Floppy disk service
0Fh     0000:003Ch      Hardware        Parallel printer service - LPT 1
10h     0000:0040h      Software        Video service routine
11h     0000:0044h      Software        Equipment list service routine
12h     0000:0048H      Software        Memory size service routine
13h     0000:004Ch      Software        Hard disk drive service
14h     0000:0050h      Software        Serial communications service routines
15h     0000:0054h      Software        System services support routines
16h     0000:0058h      Software        Keyboard support service routines
17h     0000:005Ch      Software        Parallel printer support services
18h     0000:0060h      Software        Load and run ROM BASIC
19h     0000:0064h      Software        DOS loading routine
1Ah     0000:0068h      Software        Real time clock service routines
1Bh     0000:006Ch      Software        CRTL - BREAK service routines
1Ch     0000:0070h      Software        User timer service routine
1Dh     00000074h       Software        Video control parameter table
1Eh     0000:0078h      Software        Floppy disk parameter routine
1Fh     0000:007Ch      Software        Video graphics character routine
20h-3Fh 0000:0080f-0000:00FCh SW        DOS interrupt points
40h     0000:0100h      Software        Floppy disk revector routine
41h     0000:0104h      Software        hard disk drive C: parameter table
42h     0000:0108h      Software        EGA default video driver
43h     0000:010Ch      Software        Video graphics characters
44h     0000:0110h      Software        Novel Netware API
45h     0000:0114h      Software        Not used
46h     0000:0118h      Software        Hard disk drive D: parameter table
47h     0000:011Ch -    Software        Not used
48h             Software        Not used
49h     0000:0124h      Software        Not used
4Ah     0000:0128h      Software        User alarm
4Bh-63h         0000:012Ch -    Software        Not used
64h             Software        Novel Netware IPX
65h-66h                 Software        Not used
67h             Software        EMS support routines
68h-6Fh         0000:01BCh      Software        Not used
70h     0000:01c0h      Hardware        Real time clock
71h     0000:01C4h      Hardware        Redirect interrupt cascade
72h-74h         0000:01C8h - 0000:01D0h         Hardware        Reserved - Do not use
75h     0000:01D4h      Hardware        Math coprocessor exception
76h     0000:01D8h      Hardware        Hard disk support
77h     0000:01DCh      Hardware        Suspend request
78h-79h         0000:01E0h -    Hardware        Not used
7Ah             Software        Novell Netware API
78h-FFh         0000:03FCh      Software        Not used
```

- El contenido de la tabla depende de la generación de la cpu de intel
- Primera columna: Número del vector de interrupción. Número de la entrada a la tabla de vectores.
- Segunda columna: el offset en la tabla del número de vecto de interrupción
- Columna X: Falta en la tabla.
  - El vector de 4 bytes: **Es un puntero a la rutina de atención a la interrupción ISR**
  - La dirección es segmentada. Segmento:Offset. Dos bytes para el segmento y otros dos para el offset
- Direccionamiento:
  - El Registro IDTR apunta a la primera entrada de la tabla.

  - The IVT table is typically located at 0000:0000H, and is 400H bytes in size (**4 bytes for each interrupt of 265 interruptions**).
  - Observamos que podemos obtener la dirección relativa múltiplicando el número de interrupción x4.
  - Al vector 9 le corresponde el offset IVT 36, es decir, 0x24 → en forma segmentada 0000:0024h
  - El offset de la última entrada será = 4 x 0xFF = 0x400-4 =0x3FC

- Tipos de interrupciones

  - The first 32 vectors are reserved for the processor's internal *exceptions* (0x00-0x1F)
  - Las interrupciones 0x20-0xFF son interrupciones *hadware* IRQ.
  - PIC
    - El controlador PIC es el encargado de mapear la señal IRQ a un vector de entrada a la tabla.
    - Periférico IRQ0 → PIC vector 0x20 → Tabla IVT puntero 0000:0080f (RAM) → llamada a la función ISR de atención al periférico IRQ0 (RAM)

9.6.2. Modo Protegido: Tabla IDT

- En las plataformas con S.O. una vez finalizadas las operaciones en modo real el bootloader finaliza la carga del sistema operativo y la cpu se configura en modo protegido no pudiendo el usuario: ejecutar módulos del S.O como los drivers, acceder a cualquier región de la memoria física, registros privilegiados, instrucciones privilegiadas,...
- El S.O. configura la tabla de descripción de interrupciones IDT con la misma función que la IVT pero distinto contenido.
- Interrupt Descriptor Table IDT

IDT Offset	INT #	Description
0x0000	0x00	Divide by 0
0x0004	0x01	Reserved
0x0008	0x02	NMI Interrupt
0x000C	0x03	Breakpoint (INT3)
0x0010	0x04	Overflow (INTO)
0x0014	0x05	Bounds range exceeded (BOUND)
0x0018	0x06	Invalid opcode (UD2)
0x001C	0x07	Device not available (WAIT/FWAIT)
0x0020	0x08	Double fault
0x0024	0x09	Coprocessor segment overrun
0x0028	0x0A	Invalid TSS
0x002C	0x0B	Segment not present
0x0030	0x0C	Stack-segment fault
0x0034	0x0D	General protection fault
0x0038	0x0E	Page fault
0x003C	0x0F	Reserved
0x0040	0x10	x87 FPU error
0x0044	0x11	Alignment check
0x0048	0x12	Machine check
0x004C	0x13	SIMD Floating-Point Exception
0x00xx	0x14-0x1F	Reserved
0x0xxx	0x20-0xFF	User definable → IRQ

- El contenido depende del kernel del S.O.
- Primera columna: offset a la rutina de antención a interrupción ISR

- Segunda columna: número del vector de interrupción.
- tipos de interrupción
  - 0-0x1F: *exceptions ERROR* y NMI
  - 0x20-0x2F: INT maskable: IRQ0-----IRQ15
  - 0x30-0xFF: *exceptions SW*
  - 0x80
    - isa x86-64: *syscall*
    - isa x86: *int 0x80*
- ¿A que rutina apunta el vector 0x0E? → Page Fault
- Descripción de las Entradas
  - IDTR: registro que apunta a la primera entrada de la tabla
  - Cada entrada son 8 bytes que intel llama gates.
  - Contiene un selector de segmento que identifica un descriptor de segmento de la tabla de descriptores de segmentos (ver segmentación intel)

9.6.3. IRQ

- XT-PIC interrupts use a pair of Intel 8259 programmable interrupt controllers (PIC)
  - PIC configurado por el kernel **Linux** : Understanding the Linux Kernel, Second Edition by Marco Cesati, Daniel P. Bovet
  - Example XT-PIC IRQ Assignment , intel interrupts paper: Esta configuración es un ejemplo, es decir, el SO puede **reprogramarla** y variar su configuración.

```
IRQ     Interrupt Hardware Device (vector de la tabla)
0       32 Timer
1       33 Keyboard
2       34 PIC Cascade
3       35 Second Serial Port (COM2)
4       36 First Serial Port (COM 1)
5       37 <Free>
6       38 Floppy Disk
7       39 <Free>
8       40 System Clock
9       41 <Free>
10      42 Network Interface Card(NIC)
11      43 USB Port, and Sound Card
12      44 Mouse (PS2)
13      45 Math Co-Processor
14      46 IDE Channel 1
15      47 IDE Channel 2

Note: Linux* requires IRQ 0, 2, and 13 to be as shown.
```

  - Master 8259 (PC compatible)

IVT Offset	INT #	IRQ #	Description
0x0020	0x08	0	PIT
0x0024	0x09	1	Keyboard
0x0028	0x0A	2	8259A slave controller
0x002C	0x0B	3	COM2 / COM4
0x0030	0x0C	4	COM1 / COM3
0x0034	0x0D	5	LPT2
0x0038	0x0E	6	Floppy controller
0x003C	0x0F	7	LPT1

  - Segunda columna: número de interrupción en el PIC
  - Tercera columna: número de interrupción IRQ
  - Primera columna: offset de esa entrada respecto de la primera entrada. Número de Vector.
- Slave 8259

IVT Offset	INT #	IRQ #	Description
0x01C0	0x70	8	RTC
0x01C4	0x71	9	Unassigned
0x01C8	0x72	10	Unassigned
0x01CC	0x73	11	Unassigned
0x01D0	0x74	12	Mouse controller
0x01D4	0x75	13	Math coprocessor
0x01D8	0x76	14	Hard disk controller 1
0x01DC	0x77	15	Hard disk controller 2

  - Segunda columna: número de interrupción en el PIC
  - Tercera columna: número de interrupción IRQ
  - Primera columna: offset de esa entrada respecto de la primera entrada. Número de Vector.

9.6.4. Linux

- Interrupciones configuradas por el kernel : `cat /proc/interrupts`

9.6.5. Lineas compartidas

- https://nptel.ac.in/courses/Webcourse-contents/IIT-%20Guwahati/comp_org_arc/web/module06_io/lect_03_intr/lect_03.htm
- Si una línea de interrupción está compartida por varios dispositivos, cuando uno de los dispositivos envía la señal de interrupción por la línea común, la CPU puede identificar el dispositivo que solicita la interrupción de varias maneras:
  - sondeo por software a cada miembro de la línea
  - la línea de concesión de la CPU hacia los dispositivos colocados en modo daisy-chain (se pasan la concesión entre ellos, de uno en uno), cuando la concesión llegue al miembro que solicita, este devuelve su identificación.

10. Acceso Directo a Memoria DMA

10.1. Funcionalidad

- Realizar las transferencias de datos liberando así a la CPU
- Aplicación: Transferencias de datos entre el disco duro y la memoria principal
- Unidad: DMAC (DMA Controller)
  - Puede tener varios canales DMA: cada canal se ocupa de la transferencia de un periférico.

10.2. Transferencias

- Modo ráfaga
  - Una vez que el DMAC toma el control del bus del sistema no lo cede hasta que la transferencia de todo el bloque es completada
  - Mientras el bus del sistema está ocupado por el DMAC la CPU puede operar con la memoria caché.
- Modo robo de ciclo
  - El DMAC devuelve el control del bus del sistema a la CPU cada vez que transfiere una palabra.
  - El bus es compartido en el tiempo: útil en sistemas críticos en tiempo real
- Modo transparente
  - El DMAC únicamente se adueña del bus cuando está libre y no lo necesita la CPU.

10.3. Sincronización

- La CPU puede iniciar una operación DMA en los límites del ciclo de bus de lectura o escritura. Por lo tanto se puede iniciar una operación DMA durante el ciclo de instrucción .

10.4. Operación del controlador DMA

10.4.1. Secuencia de pasos a nivel alto

- Cuando un proceso realiza una llamada *read*, el driver le asigna una región de memoria principal (DMA buffer) y genera la señales hw para solicitar la transferencia de datos al DMA buffer. El proceso queda en estado *sleep*.
- El DMAC transfiere los datos al buffer DMA y activa una señal de interrupción cuando finaliza
- El gestor de interrupciones ubica los datos del buffer al lugar definitivo, avisa de interrupción atendida y despierta al proceso, el cual ya puede leer los datos de la memoria principal.

10.4.2. Secuencia de pasos a nivel bajo

- Tres parámetros a programar:
  - dirección inicial de MPrincipal del bloque de datos a transferir: AR
  - Número de datos a transferir: WC
  - Modo de transferencia
- Pasos
  a. La *CPU* durante el arranque de la computadora inicializa el DMAC programando los parámetros.
  b. El *controlador del periférico* solicita su servicios.
  c. El *periférico* realiza una petición de DMA al DMAC (DMA Controller): *DMA Request*.
  d. El DMAC le responde con una señal de aceptación
  e. El DMAC activa la línea de petición de DMA a la *CPU*: *Bus Request*
  f. Al final del *ciclo del bus* en curso, el procesador pone las líneas del bus del sistema en alta impedancia y activa la sesión de DMA: *Bus Grant*
  g. El DMAC asume el *control del bus del sistema*

h. El dispositivo de E/S transmite una nueva palabra de datos al registro intermedio de datos del DMAC (un pequeño *buffer* en el DMAC)
i. El DMAC ejecuta un ciclo de escritura en memoria para transferir el contenido del registro intermedio a la posición M[AR].
j. El DMAC decrementa WC e incrementa AR.
k. El DMAC libera el bus y desactiva la línea de petición de DMA.
l. El DMAC compara WC con 0:
m. Si WC > 0, se repite desde el paso 2.
n. Si WC = 0, el DMAC se detiene y envía una petición de interrupción al procesador.

10.5. Problemas de coherencia en la memoria cache

- El controlador DMA al transferir datos entre el periférico y la memoria Principal provoca que las líneas de la memoria caché no sean copia de los bloques de la memoria principal. Será necesario que la controladora de la caché actualice la memoria caché después de una operación DMA.

11. Buses

- La arquitectura i/o ha ido evolucionando en dos direcciones
  - incremento del ancho de banda de los buses
  - integración de los controladores i/o en un único chip

11.1. ISA

./images/io/isa_8086.png

Figura 2: Arquitectura Bus ISA

./images/io/isa_8086_interfaz.png

Figura 3: Interfaz Bus ISA

11.2. PCI

./images/io/pci_80x86.png

Figura 4: Arquitectura Bus PCI

11.3. North-South Bridge

./images/io/north_south_bridge.png

Figura 5: North and South Bridges

11.4. Chipset x58

- https://en.wikipedia.org/wiki/Intel_X58
- PCH: Platform Controller Hub
- FSB: Front Side Bus
- BSB: Back Side Bus
- FDI: Flexible Display Interface (para CPU que integran la controladora gráfica)
- DMI: Direct Media Interface
- ICH: i/o Controller Hub
- IOH: i/o Hub

./images/io/x58blockdiagram_corei7.jpg

Figura 6: corei7 x58 chipset: año 2008

Chipsets supporting LGA 1366, LGA 2011, and LGA 2011-v3 CPUs.: X58 (2008), X79 (2011), X99 (2014).

12. Programacion de rutinas de entrada/salida

12.1. Software jerarquico del sistema operativo

- El driver o controlador sw es el nivel más bajo de la estructura sw: depende fuertemente del hardware de la computadora: programación en lenguaje C o ensamblador.

12.2. Instruction Set Architecture

- I/O access
  - OUTx : Sends a byte (or word or dword) on a I/O location. Traditional names are *outb*, *outw* and *outl* respectively. The "a" modifier enforces *val* to be placed in the eax register before the asm command is issued and "Nd" allows for one-byte constant values to be assembled as constants, freeing the edx register for other cases.

```
static inline
void outb( unsigned short port, unsigned char val )
{
    asm volatile( "outb %0, %1"
                  : : "a"(val), "Nd"(port) );
}
```

    - El programa fuente en C incluye lenguaje ASM: Programa fuente en C con inline-asm.
    - %0 hace referencia a la primera variable "a", %i hace referencia a la i-nésima variable.
  - INx : Receives a byte (or word or dword) from an I/O location. Traditional names are *inb*, *inw* and *inl* respectively.

```
static inline
unsigned char inb( unsigned short port )
{
    unsigned char ret;
    asm volatile( "inb %1, %0"
                  : "=a"(ret) : "Nd"(port) );
    return ret;
}
```

- The register I/O instructions IN (input from I/O port) and OUT (output to I/O port) move data between I/O ports and the EAX register (32-bit I/O), the AX register (16-bit I/O), or the AL (8-bit I/O) register. The address of the I/O port can be given with an immediate value or a value in the DX register.

12.2.1. Intel Manual

- This instruction is only useful for accessing I/O ports located in the processor's I/O address space. See Chapter 16 or 14, "Input/Output," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for more infor-mation on accessing I/O ports in the I/O address space.
- I/O ports can be mapped so that they appear in the I/O address space or the physical-memory address space (memory mapped I/O) or both.
- **memory-mapped**:
  - mediante una línea del bus de control se especifica si la dirección es de memoria principal o port i/o, en algún procesador mediante el M/IO# pin.
  - When using memory-mapped I/O, caching of the address space mapped for I/O operations must be prevented
- **I/O mapped**
  - i/o devices don't collide with memory, as they use a different *address space*, with different instructions to read and write values to addresses (ports). CPU decode the memory-I/O bus transaction instructions to select I/O ports These devices cannot be addressed using machine code instructions that targets memory. What is happening is that there are two different signals: *MREQ* and *IOREQ*. The first one is asserted on every memory instruction, the second one, on every I/O instruction. So this code...

```
MOV DX,1234h
MOV AL,[DX]     ;reads memory address 1234h (memory address space)
IN AL,DX        ;reads I/O port 1234h (I/O address space)
```

  - The I/O device at port 1234h is connected to the system bus so that it is enabled only if the address is 1234h, RD (Read Data) is asserted and IOREQ is asserted.
  - (64K) individually addressable 8-bit I/O ports
- **Protection**
  - Port Mapped
    - Here, kernel and the device drivers are allowed to perform I/O, while less privileged device drivers and application programs are denied access to the I/O address space. Application programs must then make *calls* to the operating system to perform I/O.
  - Memory mapped
    - the normal segmentation and paging protection affect the i/o port access.

12.3. Programación del Controlador de Interrupciones Programable

- programación del pic
  - Mapeo del PIC

```
/* remap the PIC controller interrupts to our vectors
   rather than the 8 + 70 as mapped by default */

#define PIC1            0x20
#define PIC2            0xA0
#define PIC1_COMMAND    PIC1
#define PIC1_DATA       (PIC1+1)
#define PIC2_COMMAND    PIC2
```

```
#define PIC2_DATA       (PIC2+1)
#define PIC_EOI         0x20

#define ICW1_ICW4       0x01            /* ICW4 (not) needed */
#define ICW1_SINGLE     0x02            /* Single (cascade) mode */
#define ICW1_INTERVAL4  0x04            /* Call address interval 4 (8) */
#define ICW1_LEVEL      0x08            /* Level triggered (edge) mode */
#define ICW1_INIT       0x10            /* Initialization - required! */

#define ICW4_8086       0x01            /* 8086/88 (MCS-80/85) mode */
#define ICW4_AUTO       0x02            /* Auto (normal) EOI */
#define ICW4_BUF_SLAVE  0x08            /* Buffered mode/slave */
#define ICW4_BUF_MASTER 0x0C            /* Buffered mode/master */
#define ICW4_SFNM       0x10            /* Special fully nested (not) */

void remap_pics(int pic1, int pic2)
{
        UCHAR   a1, a2;

        a1=inb(PIC1_DATA);
        a2=inb(PIC2_DATA);

        outb(PIC1_COMMAND, ICW1_INIT+ICW1_ICW4);
        io_wait();
        outb(PIC2_COMMAND, ICW1_INIT+ICW1_ICW4);
        io_wait();
        outb(PIC1_DATA, pic1);
        io_wait();
        outb(PIC2_DATA, pic2);
        io_wait();
        outb(PIC1_DATA, 4);
        io_wait();
        outb(PIC2_DATA, 2);
        io_wait();

        outb(PIC1_DATA, ICW4_8086);
        io_wait();
        outb(PIC2_DATA, ICW4_8086);
        io_wait();

        outb(PIC1_DATA, a1);
        outb(PIC2_DATA, a2);
}
```

12.4. Driver del Teclado

- Fijarse cómo se programa teniendo en cuenta el mecanismo de atención a las interrupciones.
- Buscar el código fuente de un kernel sencillo.

12.5. paralell port

12.5.1. Desde Espacio de Usuario

- Es necesario realizar llamadas al sistema ya que no podemos acceder desde el espacio de usuario directamente al HW
- Direcciones base de los puertos
- Acceso a un puerto en linux desde el espacio de usuario

```c
/* led_bloq_mayus.c: very simple example of port I/O
 *
 * This code active LED keyboard CAP, just a port write, a pause,
 * and a port read. Compile with `gcc -O2 -o led_bloq_mayus led_bloq_mayus.c',
 * and run as root with `sudo ./led_bloq_mayus'.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/io.h>

#define BASEPORT 0x0060 /* keyboard */

int main()
{
        /* Get access to the ports */
        if (ioperm(BASEPORT, 3, 1)) {perror("ioperm"); exit(1);}
        printf("\n\t\t Port -> registro status: %d\n", inb(BASEPORT + 1));

        /* Set the data signals (D0-7) of the port to all low (0) */
        outb(0xED, BASEPORT);

        /* Sleep for a while (100 ms) */
        usleep(1000);
        printf("\n \t\tActiva el LED de la tecla BLOQ MAYUS \n ");
        outb(0x07, BASEPORT);

        usleep(1000);

        /* We don't need the ports anymore */
        if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}
    exit(0);
}
```

- http://tldp.org/HOWTO/IO-Port-Programming-2.html
- man ioperm
- man inb
- `cat /proc/ioports`

- http://opensourceforu.efytimes.com/2011/07/accessing-x86-specific-io-mapped-hardware-in-linux/

12.6. Serial communication RS-232

- tutorial
  - Tarjeta Avr Atmega 8bits → Pej Arduino One
  - Dos casos: Polling i/o e interrupt-driven i/o
- Puerto UART (RS-232)
  - Conector físico
  - Comunicación semiduplex entre dos terminales: DTE (PC) y DCE (Arduino)
  - Señales Tx Rx
  - Registros del puerto
    - Control

    - Estado
    - Datos Rx y Tx: UDR

- Programación
  - Librería
  - Cross toolchain
  - Algoritmo: Diagrama de flujo
    - Casos Polling y Interruption
  - Estructura modular: dos módulos
  - Símbolos
    - Buffer de datos i/o
    - Nombre del vector de interrupción

13. Ejercicios

- Capítulo 7 del libro de texto William Stalling.