Arquitectura del Computador
y
Sistemas Operativos

Novena Clase

Dispositivos de E/S (1/3)

Toda CPU necesita una interfaz con el mundo real. Esta interfaz puede ser con un ser humano, en cuyo caso podríamos pensar en monitor, teclado y/o mouse, o no. Imaginen una computadora que corre un software de modelos climáticos, cuyo objetivo es predecir el clima en una zona. Seguramente necesitaría datos de anemómetros, higrómetros, termómetros, y datos satelitales. Podría emitir la salida imprimiendo un reporte.

Toda pieza de hardware conectada al computador que le permite obtener datos del mundo exterior lo llamamos "*Dispositivo de Entrada*" y los que presentan datos al mundo exterior "Dispositivo de Salida".

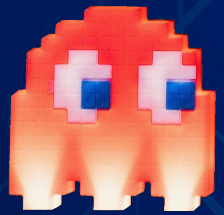
Son componentes tan importantes que forman parte de las primeras computadoras que se han diseñado, tal como vimos al analizar la arquitectura propuesta por Von Neumann.

Dispositivos de E/S (2/3)

Todo dispositivo de entrada o salida, desde el punto de vista del procesador, es una (o más) posiciones de memoria.

Ejemplos:

- Un teclado tiene sus teclas codificadas. Cada vez que presionan una tecla, un valor numérico (llamado scan-code) es colocado en una dirección de memoria que el procesador lee.



- Un monitor puede verse como miles de puntos. Cada punto recibe su color de 3 bytes. Cada byte contiene el nivel de un color (RGB). Cuando el procesador modifica los valores en esa zona de memoria el punto asociado cambia de color.

Definitivamente no es óptimo tener que estar consultando una y otra vez una posición de memoria para ver si un evento ocurrió.

Dispositivos de E/S (3/3)

El teclado PS/2 se ve desde el procesador como dos posiciones de memoria.

Si el bit DR=1 quiere decir que el procesador tiene información para leer, y puede hacerlo leyendo la posición de memoria 0x60. Los bits PE y TE le dicen si el valor recibido del teclado es válido o la transmisión sufrió errores.

Leer del Data Register baja DR a 0.

El bit CB indica si el teclado está en condiciones de recibir información del procesador. Éste debe esperar que CB=0 antes de mandarle datos al teclado, cosa que hace escribiendo en el Data Register.

Control Register - I/O 0x64

7	6	5	4	3	2	1	0
PE	TE					CB	DR

DR: Data ready
CB: Controller Busy
TE: Timeout Error
PE: Parity Error

Data register - I/O 0x60

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

Interrupciones (1/4)

Los procesadores contienen un mecanismo llamado "*interrupción*". Funciona así:

- El procesador tiene una entrada digital que un dispositivo externo puede activar para generar la interrupción en la ejecución del código.
- Al terminar de ejecutar cada instrucción, el procesador verifica si esa entrada está activa.
- Si es así:
  - Coloca en el stack el valor del PC (PUSH PC)
  - Coloca en el stack el valor del registro Flags (PUSHF)
  - Reemplaza el valor del PC por una dirección que el software definió con anterioridad
  - El control sigue ejecutando en la nueva dirección
- El código ahora puede analizar por qué el dispositivo externo interrumpió y tomar las acciones adecuadas.
- Al terminar ejecuta la instrucción IRET, que equivale a:
  - Saca del stack el valor del registro Flags (POPF)
  - Saca del stack el PC (POP PC)
- La ejecución continúa justo donde se había interrumpido.

Interrupciones (2/4)

A la dirección a la que se debe saltar al recibir la interrupción se la denomina “vector de interrupción”.

La rutina que se ejecuta se llama “*Interrupt Service Routine*” (ISR).

Que una interrupción entre mientras se ejecuta la ISR puede llevar a resultados indeseados. Por éso los procesadores tienen un bit en el registro Flags que se denomina “Interrupt Mask”. Se la puede poner en 1 con la instrucción STI o en 0 usando CLI. Mientras vale 1 el procesador busca interrupciones al final de cada instrucción, cuando vale 0 sigue con la siguiente instrucción sin verificar.

Uno de los pasos realizados al detectar una interrupción (justo después de pasar los Flags al stack), se baja a 0 el bit de “*Interrupt Mask*”. El código de la ISR ahora ejecuta sin que pueda entrar otra interrupción. Al ejecutar IRET, cuando los Flags se restauran con la copia que se dejó en el stack, el valor vuelve a 1 y puede entrar una nueva interrupción.

Interrupciones (3/4)

Los procesadores más pequeños suelen tener dos líneas de interrupción, la IRQ (que vimos antes) y la "*Non Maskable Interrupt*", o NMI. Esta última no puede enmascararse. Cada una tiene su propio vector de interrupción.

Los procesadores más complejos, tienen un chip externo dedicado al manejo de interrupciones que maneja decenas de ellas. A cada una se le asigna una prioridad, por lo que hay algunas que pueden interrumpir a otras.

El mecanismo que vimos, no solo se usa para interrupciones externas. Hay varias situaciones internas que generan interrupciones al procesador, entre ellas:

- División por cero
- OP Code inválido

En el 8086 hay 256 posibles interrupciones, la tabla que contiene las direcciones a la cual saltar al ocurrir cada una se denomina "*Interrupt Descriptor Table*", o IDT.

Interrupciones (4/4)

Las interrupciones también se pueden generar mediante una instrucción. Por ejemplo:

```
    .data
    MiMensaje     DB        “Hola Mundo”, 0x10
    .code
                  MOV       EAX,4                       # sys_write
                  MOV       EBX,1                       # 1=stdout
                  MOV       ECX, BYTE PTR MiMensaje     # Mensaje
                  MOV       EDX,11                      # Longitud texto
                  INT       80h
```

Estas interrupciones se denominan “*Software Interrupts*”, y son la forma de acceder a las funciones muchos Sistemas Operativos de 16 o 32 bits.

Para los Sistemas Operativos de 64 bits se usa un nuevo opcode denominado SYSCALL

Veremos más adelante que el proceso solicitarle al SO una acción se denomina “*System Call*”.

Direct Memory Access (DMA) (1/2)

Para dispositivos como los discos, a los que se les pide, por ejemplo, levantar un sector 512 bytes, resulta terriblemente ineficiente que interrumpa por cada byte que lee.

Es mucho más óptimo alocarle una zona de memoria y decirle que levante el sector completo e interrumpa cuando terminó. Ésto involucra **Compartir los Buses de Address y Datos**.

Parecería que si ahora somos dos los que usamos la memoria, la performance también se vería degradada, dado que podría darse el caso que los dos (CPU y Dispositivo Externo) necesitáramos acceder a la memoria, pero hay formas de que el impacto sea mínimo.

A este proceso se lo conoce como DMA, dado que el dispositivo no necesita al procesador para acceder a memoria.

El ejemplo dado para lectura aplica también a escritura. El procesador puede darle al dispositivo externo un puntero a una zona de memoria de la cual sacar todo el sector y que interrumpa cuando ya está en disco.

Direct Memory Access (DMA) (2/2)

Los ordenadores que soportan DMA tienen hardware especifico para ese fin. Suelen tener varios canales para que puedan usarlos varios dispositivos a la vez.

Este hardware es configurado por el software indicando si la transferencia es “Dispositivo a Memoria” o “Memoria a Dispositivo”. También se le carga la dirección de inicio del bloque de memoria y la longitud del mismo.

Una vez configurado el canal de DMA, el software le indica al Dispositivo Externo que puede iniciar la transferencia. A partir de ese momento, dicho dispositivo simplemente pulsa una línea reiteradas veces. Por cada pulso se transfiere una palabra.

Fin

¿Preguntas?