Entrada / Salida
Ampliación

Agustín Fernández, Josep Llosa, Fermín Sánchez

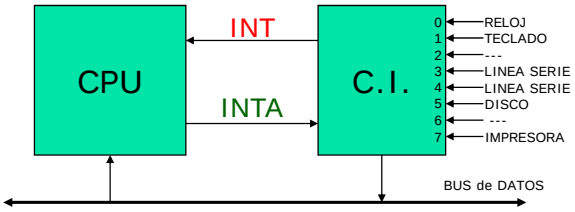
Estructura de Computadors II
Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona

Guión

- Sincronización por interrupciones (ejemplo real: IBM-PC)
- Ejemplos de programación
- DMA y Jerarquía de Memoria
- Dispositivos

Sincronización por Interrupciones PC compatible

- 1) Detección de la petición de interrupción: Controlador de Interrupciones.

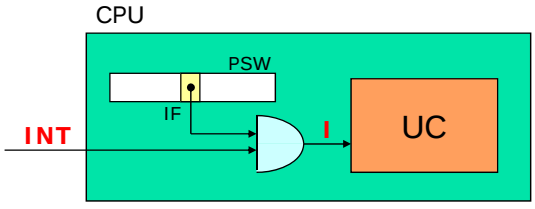


El controlador de interrupciones:

- Establece prioridades.
- Mantiene peticiones pendientes.
- Permite inhibir interrupciones selectivamente.

Sincronización por Interrupciones PC compatible

- 1) Detección de la petición de interrupción: Controlador de Interrupciones.

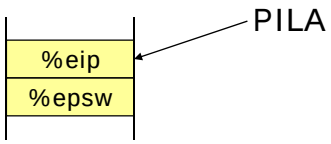


El procesador dispone de 2 instrucciones de LM:

- STI: IF <- 1 // enable();
- CLI: IF <- 0 // disable();

Sincronización por Interrupciones PC compatible

- 2) Salvar estado del programa interrumpido
  - Salva en la pila el contador de programa y la palabra de estado.



  - Además, inhibe interrupciones (IF=0) → no se aceptan más interrupciones hasta que se acabe de servir la interrupción o bien se ejecute una instrucción STI en la RAI.

Sincronización por Interrupciones PC compatible

- 3) Identificar la rutina a ejecutar
  - Cuando el Controlador de Interrupciones recibe el INTA, le envía al procesador un número (N) que identifica el dispositivo que ha interrumpido (p.e. para el RELOJ N=8, TECLADO N=9, ...).
  - La dirección de la RAI se encuentra en la entrada N del vector de interrupciones. El vector de interrupciones está almacenado a partir de la dirección 0 de memoria. En definitiva, la dirección de la RAI es:

$$\%eip \leftarrow Mem[N*4]$$

Sincronización por Interrupciones PC compatible

- 4) Ejecutar la rutina de atención a la interrupción (RAI)

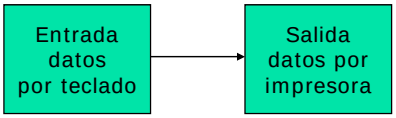
- 5) Retorno al programa interrumpido
  - Antes de acabar la RAI, hay que avisar al Controlador de Interrupciones de que hemos acabado. Hay que enviar un comando EOI().
    - En el PC: escribir 0x20 en el registro de control del Controlador de Interrupciones que se encuentra en el puerto de E/S 0x20.
  - Para retornar al programa interrumpido se usa una instrucción de retorno especial:

    IRET

Ejemplo de Programación: Máquina de Escribir

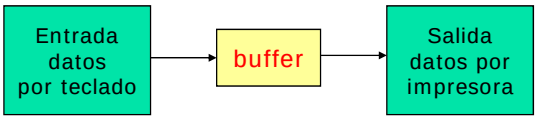
- Usaremos como ejemplo los dispositivos SISA-F:
  - Teclado:
    - Rcon_tec: bit0=1, funcionamiento por interrupciones
    - Rest_tec: bit0=1, tecla pulsada
    - Rdat_tec: código de rastreo
  - Impresora:
    - Rdat_imp: carácter a imprimir
    - Rcon_imp: bit0=1, funcionamiento por interrupciones; bit15=1, orden de imprimir carácter
    - Rest_imp: bit0=1, impresora preparada.
  - Controlador Interrupciones (tipo PC), teclado por la IRQ1, impresora por la IRQ7.
    - Al acabar una interrupción hay que ejecutar el comando “EOI()”.

Ejemplo de Programación: Máquina de Escribir



¡Problema: funcionará a la velocidad del dispositivo más lento!

Solución: utilizar un buffer intermedio.

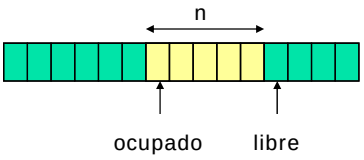


Ejemplo de Programación: Máquina de Escribir

Descripción del buffer.

```
typedef struct buffer {
   char v[N];
   int libre, ocupado, n;
}
void InitB(buffer *b) {
  (*b).libre = 0;
  (*b).ocupado = 0;
  (*b).n = 0;
}
int VacioB(buffer *b) {
  return (*b).n == 0;
}
int LlenoB(buffer *b) {
  return (*b).n == N;
}
```

```
void PutB(buffer *b, char c) {
  (*b).v[(*b).libre] = c;
  (*b).libre = ((*b).libre + 1) % N
  (*b).n++;
}
char GetB(buffer *b){
  char c;
  c = (*b).v[(*b).ocupado];
  (*b).ocupado = ((*b).ocupado+1)%N;
  (*b).n--;
  return c;
}
```



1a solución: Teclado e impresora por encuesta

```
main() {
  buffer B;
  . . .
  InitB(&B);
  x = in(Rcon_imp);
  x = x & 0xfffe;
  out(Rcon_imp, x);
  x = in(Rcon_tec);
  x = x & 0xfffe;
  out(Rcon_tec, x);
```

```
  for (;;) {
     if (in(Rest_tec) & 0x0001)
       if (! LlenoB(&B)) {
         c = in(Rdat_tec);
         PutB(&B, c);
       }
     if (in(Rest_imp) & 0x0001)
       if (! vacioB(&B)) {
         c = GetB(&B);
         out(Rdat_imp, c);
          x = in(Rcon_imp);
          x = x | 0x8000;
          out(Rcon_imp, x);
       }
   }
}
```


1a solución: Teclado e impresora por encuesta

```
main() {
  char v[N];

  libre=0;
  ocupado=0;
  n=0;
  x = in(Rcon_imp);
  x = x & 0xfffe;
  out(Rcon_imp, x);
  x = in(Rcon_tec);
  x = x & 0xfffe;
  out(Rcon_tec, x);
```

```
for (;;) {
  if (in(Rest_tec) & 0x0001)
     if (n<N) {
       v[libre] = in(Rdat_tec);
       libre = (libre+1)%N;
       n++;
     }
  if (in(Rest_imp) & 0x0001)
     if (n>0) {
       out(Rdat_imp, v[ocupado]);
       x = in(Rcon_imp);
       x = x | 0x8000;
       out(Rcon_imp, x);
       ocupado = (ocupado+1)%N;
       n--;
     }
  }
}
```


2a solución: Teclado por interrupciones e impresora por encuesta

```
main() {
  buffer B;
  . . .
  InitB(&B);
  x = in(Rcon_imp);
  x = x & 0xfffe;
  out(Rcon_imp, x);
  x = in(Rcon_tec);
  x = x | 0x0001;
  out(Rcon_tec, x);
  disable()
  setvect(9, (void interrupt(*)()) teclado);
  enable();

  for (;;)
    if (in(Rest_imp) & 0x0001)
      if (! vacioB(&B)) {
        c = GetB(&B);
        out(Rdat_imp, c);
         x = in(Rcon_imp);
         x = x | 0x8000;
         out(Rcon_imp, x);
      }

}
```

```
void interrupt teclado()
{
  c = in(Rdat_tec);
  if (! LlenoB(&B))
    PutB(&B, c);
  EOI();
}
```


3a solución: Teclado por encuesta e impresora por interrupciones

```
main() {

  // Inicializaciones

  impresora = parada
  for (;;) {
    if (in(Rest_tec) & 0x0001)
      if (! LlenoB(&B)) {
        c = in(Rdat_tec);
        PutB(&B, c);
      }
    if ((impresora == parada) &&
       (! vacioB(&B)) &&
       (in(Rest_imp) & 0x0001) {
         c = GetB(&B);
         out(Rdat_imp, c);
         x = in(Rcon_imp);
         x = x | 0x8000;
         out(Rcon_imp, x);
         impresora = funcionando;
    }
  }
}
```

```
void interrupt impresora()
{
  if (! vacioB(&B)
    c = GetB(&B);
    out(Rdat_imp, c);
    x = in(Rcon_imp);
    x = x | 0x8000;
    out(Rcon_imp, x);
  else
    impresora = parada;
  EOI();
}
```

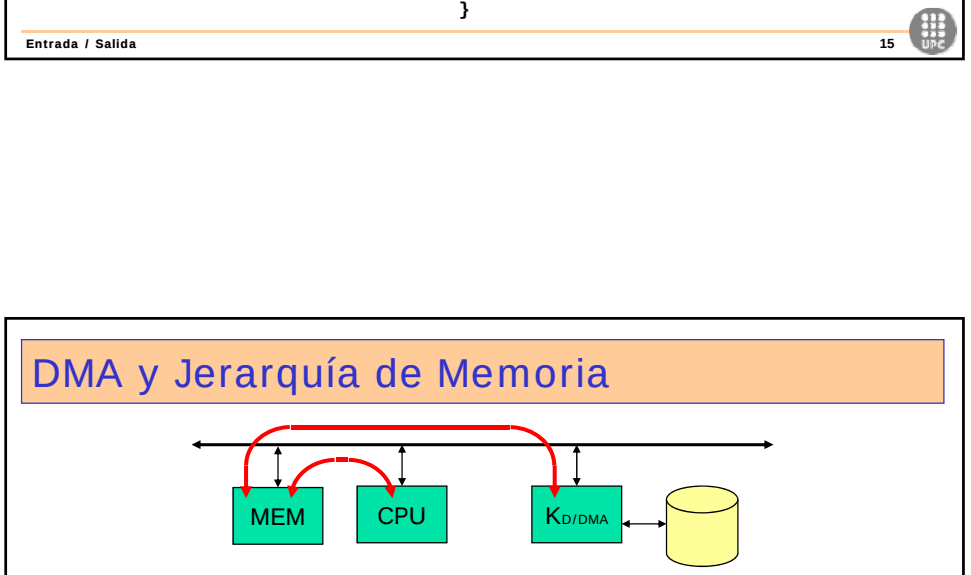

4a solución: Teclado e impresora por interrupciones

```
main() {

  Inicializaciones:
  - vector interrupciones
  - programar dispositivos

  impresora = parada
  for (;;) {
    ¡ TAREA INDEPENDIENTE !
  }
```

```
void interrupt impresora()
{ if (! vacioB(&B)
    c = GetB(&B);
    out(Rdat_imp, c);
    x = in(Rcon_imp);
    x = x | 0x8000;
    out(Rcon_imp, x);
  else
    impresora = parada;
  EOI();
}
void interrupt teclado()
{ c = in(Rdat_tec);
  if (! LlenoB(&B)) PutB(&B, c);
  if ((impresora == parada) &&
      (in(Rest_imp) & 0x0001) {
         c = GetB(&B);
         out(Rdat_imp, c);
         x = in(Rcon_imp) | 0x8000;
         out(Rcon_imp, x);
         impresora = funcionando;
    }
  EOI();
}
```

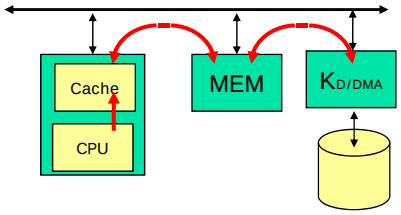
DMA y Jerarquía de Memoria

- En un computador, tanto el procesador como el K_{DMA} pueden acceder a Memoria.
- Esta situación puede provocar:
  - Problemas de COHERENCIA con la Memoria Cache
  - Problemas con la TRADUCCIÓN de DIRECCIONES

DMA y Jerarquía de Memorias

Problema de COHERENCIA 1

1. La CPU escribe un dato X en la cache.
2. Una vez escrito, el dato sigue permaneciendo en cache hasta que la línea sea substituida.
3. Se programa una escritura de disco y se almacena la posición donde estaba X (X tiene un valor distinto en Memoria Principal con lo que escribimos un VALOR INCORRECTO).

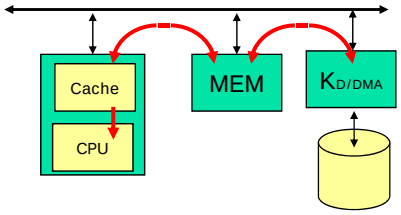


Existen diversas soluciones para este problema:
- Usar una cache **Write Trough**
- Que el K_{DMA} sólo pueda acceder a zonas de MP NO-CACHEABLES
- Vaciar la cache cada vez que se lanza una operación con el K_{DMA}

DMA y Jerarquía de Memorias

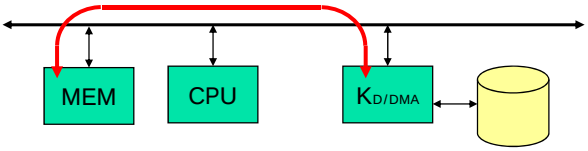
Problema de COHERENCIA 2

1. La CPU accede a un dato X y lo trae a la cache.
2. Una vez leído el dato, el dato sigue permaneciendo en cache hasta que la línea sea substituida.
3. Se programa una lectura de disco y se almacena en la posición donde estaba X (X toma un nuevo valor en Memoria Principal).
4. El procesador accede de nuevo a X, pero obtiene el valor almacenado en la cache y NO el VALOR CORRECTO que está en Memoria Principal.
   - El problema se da incluso con una cache write through



Existen diversas soluciones para este problema:
- Que el K_{DMA} sólo pueda acceder a zonas de MP NO-CACHEABLES
- Vaciar la cache cada vez que se lanza una operación con el K_{DMA}

DMA y Jerarquía de Memorias



Problemas con la TRADUCCIÓN de DIRECCIONES.

- El K_{DMA} ha de acceder a Memoria Principal utilizando direcciones físicas.
- Eso implica que cuando el SO quiera hacer una operación con el K_{DMA}, tendrá que acceder a la tabla de páginas y programarlo con direcciones físicas.