Entrada / Salida
Conocimientos Previos

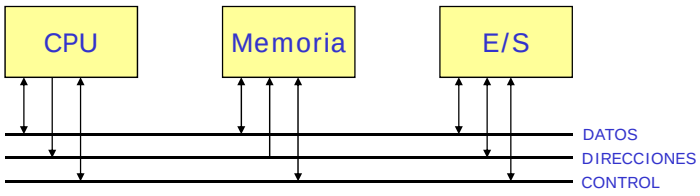
Agustín Fernández, Josep Llosa, Fermín Sánchez

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

Guión

- Controladores
- Sincronización por encuesta
- Sincronización por interrupciones
- Transferencia vía DMA

Computador básico y controladores

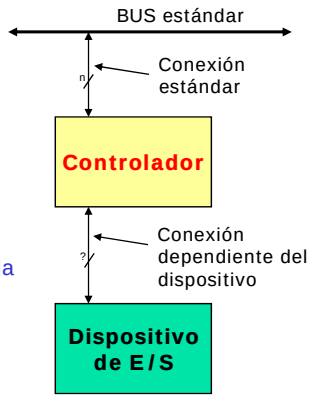


- La forma en que se gestionan los dispositivos de E/S es muy diversa.
- Si la CPU tuviera que gestionar directamente todos los dispositivos, los programas de control serían muy complejos.
- Las necesidades de interconexión de los diferentes dispositivos son muy diversas. Además hay que conectarlos a la placa base.

→ **CONTROLADORES**

Controladores

- Un controlador es un dispositivo electrónico que se interpone entre el dispositivo de E/S y los buses de la placa base.
- Las funciones del dispositivo de E/S se activan a través del controlador.
- Funciones del controlador:
  - Dialogar con la CPU (recibe peticiones, envía avisos).
  - Controlar el periférico para que realice la función solicitada por la CPU.
  - Facilitar la transferencia de información entre la CPU y el periférico.



Registros de los Controladores

- Las funciones del controlador se activan a través de los registros de E/S.
- Los controladores disponen de un conjunto de registros visibles desde LM. La CPU puede leer estos registros en cualquier momento:
  - Registro de CONTROL: indica cómo ha de trabajar el periférico, qué operación ha de realizar,...
  - Registro de ESTADO: mantiene información respecto a la situación actual del periférico (ejemplo impresora: apagada, no hay papel, no hay tinta,...).
  - Registro de DATOS: el controlador deja aquí los datos de entrada (de un teclado p.e.) o los datos que van a ser enviados al exterior (a una impresora p.e.), o los parámetros para realizar una determinada operación (cara, pista, ... en un disco).

Registros de los Controladores

- ¿Cómo se accede a los registros de los controladores?
- Registros mapeados en Memoria. Determinadas direcciones de memoria almacenan el valor de los registros de los controladores de E/S (p.e. la memoria de pantalla). Para acceder a los registros del controlador se utiliza cualquier instrucción de lenguaje máquina que permita acceder a memoria.

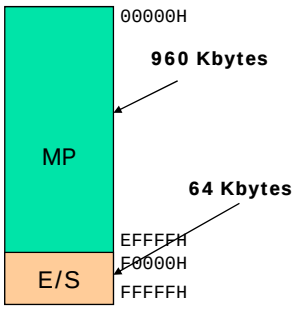
```
movw $12, (%edx,%esi)
movw (%edx), %cx
```

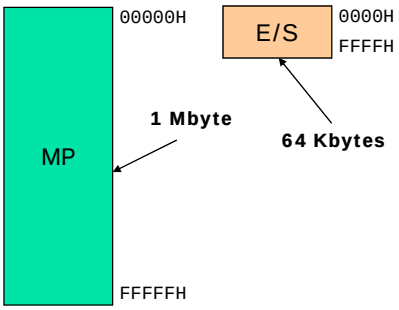
- Registros no-mapeados en Memoria. Existen instrucciones especiales para acceder a los registros de los controladores de E/S.

```
outw %ax, %dx
inw %dx, %ax
```


Registros mapeados vs no-mapeados

- Registros mapeados en Memoria.
- Registros no-mapeados en Memoria.

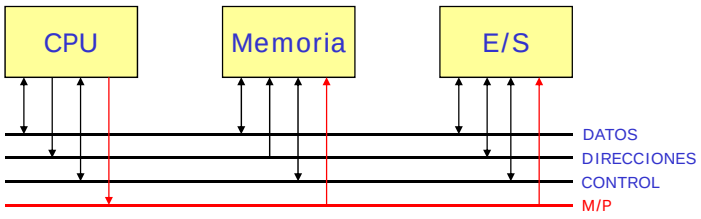




Registros mapeados vs no-mapeados

- Ventajas / Inconvenientes.
- Registros mapeados:
  - Se puede usar cualquier instrucción de LM.
  - Se pierde espacio para memoria física.
  - Es difícil controlar que un usuario no autorizado acceda a dispositivos de E/S.
- Registros no-mapeados:
  - No se desperdicia espacio de memoria.
  - Es fácil impedir que un usuario no autorizado acceda a dispositivos de E/S.
  - Es necesario definir 2 nuevas instrucciones de LM.

Registros mapeados vs no-mapeados



- Implementación:
  - Mapeados en Memoria activando la señal M/P a 0.
  - No-mapeados en Memoria activando la señal M/P a 1.

Sincronización en las Operaciones de E/S

- La sincronización de la CPU con los dispositivos de E/S es necesaria porque:
  - Si se ha de leer de un dispositivo, se ha de garantizar que el dato leído es válido (ej: un usuario no teclea siempre con la misma cadencia).
  - Si se ha de enviar un dato a un dispositivo, se ha de garantizar que el dispositivo está listo para recibir el dato.
- Existen dos formas de realizar la sincronización:
  - Sincronización por encuesta (*polling*). El procesador, mediante un programa que lee repetidamente el registro de estado del controlador, se encarga de averiguar cuándo se puede realizar la operación de E/S.
  - Sincronización por interrupciones. El controlador avisa al procesador, mediante un mecanismo hardware, cuándo se puede realizar la operación de E/S.

Sincronización por encuesta

- El procesador ha de detectar, por software, la disponibilidad del dispositivo mediante la consulta del registro de estado:

```
leer estado dispositivo;
while (dispositivo no disponible)
   leer estado dispositivo;
realizar la operación de E/S;
```

Mientras se realiza la encuesta, el procesador no realiza ningún trabajo útil.

```
for (;;){
  TAREA INDEPENDIENTE;
  if (dispositivo preparado)
  realizar la operación de E/S;
}
```


Dispositivos en SISA-F

- Pantalla (sólo de salida):
  - 16 líneas, 64 caracteres por línea
  - 4 registros de E/S:
    - Rfil_pant: fila
    - Rcol_pant: columna
    - Rdat_pant: carácter a visualizar (normal o inverso)
    - Rcon_pant: puesta en marcha
- Teclado:
  - 43 teclas: a..z 0..9 , . - : + <enter> <del>
  - 3 registros de E/S:
    - Rcon_tec: encuesta o interrupciones
    - Rest_tec: ¿Tecla pulsada?
    - Rdat_tec: código de rastreo
- Impresora:
  - 64 caracteres por línea, salta de línea con "\n"
  - 3 registros de E/S:
    - Rdat_imp: carácter a imprimir
    - Rcon_imp: encuesta o interrupción, puesta en marcha
    - Rest_imp: ¿impresora preparada?

Ejemplo E/S

- Imprimir un vector de caracteres utilizando sincronización por **encuesta**

```
void print (int n, char vec[])
{
  int i;
  int estado;

  out(Rcon_imp, 0x0000);  //La impresora funciona por encuesta
  for (i=0; i<n; n++)
  {
    do {
      estado = in(Rest_imp);
    } while (estado == 0);
    out(Rdat_imp, vec[i]);  // Enviar dato
    out(Rcon_imp, 0x8000);  // Orden de imprimir
  }
}
```


Ejemplo E/S

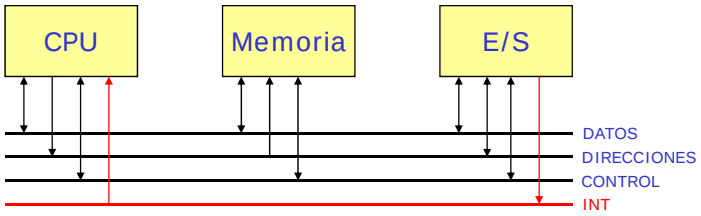
- Volcar por pantalla un vector de caracteres (no necesita sincronización)

```
void display (int n, char vec[], int fil, int col)
{
  int i;

  for (i=0; i<n; n++)
  {
    out(Rfil_pant, fil);     // fila
    out(Rcol_pant, col);     // columna
    out(Rdat_pant, vec[i]);  // Enviar dato
    out(Rcon_pant, 0x8000);  // Puesta en marcha
    col++;
  }
}
```

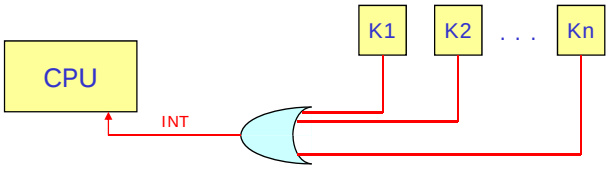

Sincronización por interrupciones

- El controlador de dispositivo avisa al procesador de que se puede realizar una operación de E/S.



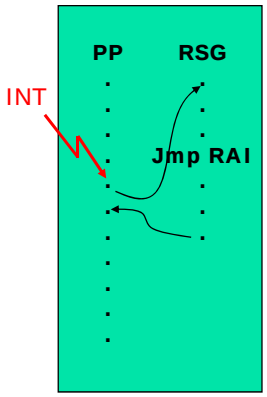
Sincronización por interrupciones

- Implementación más sencilla: Si hay más de un dispositivo que puede interrumpir al procesador, simplemente hay que hacer una OR de todas las posibles peticiones de interrupción.



Sincronización por interrupciones

- Rutina de Servicio General (RSG) y Rutina de Atención a la Interrupción (RAI).



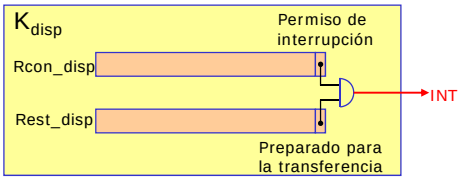
Después de activarse la señal INT, el procesador suspende la ejecución del programa y salta a ejecutar una rutina especial encargada de atender la petición de interrupción.
RSG: Rutina de Servicio General

Sincronización por interrupciones

- Fases del mecanismo de sincronización por interrupciones
  1. Generación de la petición
  2. Detección de la petición
  3. Aceptación de la petición
  4. Salvar contexto y saltar a la RSG
  5. Salvar estado
  6. Identificar dispositivo
  7. Eliminar la petición de interrupción
  8. Ejecutar la RAI específica
  9. Restaurar estado
  10. Retornar al programa interrumpido

Sincronización por interrupciones

- Fases del mecanismo de sincronización por interrupciones
  1. Generación de la petición



  2. Detección de la petición
     - Se modifica el autómata de la Unidad de Control de Procesador

```
while (true) {
  FetchInstrucción;
  Decodificación;
  LeerOperandos;
  Ejecución;
  EscribirResultado
}
```

→

```
while (true) {
  if (INT==1) TratarInterrupción;
  FetchInstrucción;
  Decodificación;
  LeerOperandos;
  Ejecución;
  EscribirResultado
}
```


Sincronización por interrupciones

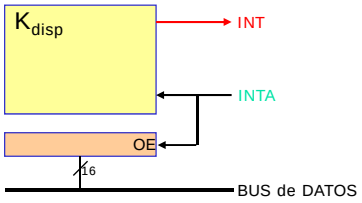
- Fases del mecanismo de sincronización por interrupciones
  3. Aceptación de la petición
     - Para aceptar la petición es necesario que las interrupciones estén permitidas (bit I = 1).

```
ei();   //permitir interrupciones
di();   //inhibir interrupciones
```

  4. Salvar contexto y saltar a la RSG
     - Salvar PC en el registro S1, la palabra de estado en S0 y el evento que causa la interrupción en S2.
     - En el PC se pone la dirección de la RSG
  5. Salvar estado
     - Lo primero que hace la RSG es salvar los registros del procesador (entre ellos S0, S1 y S2)

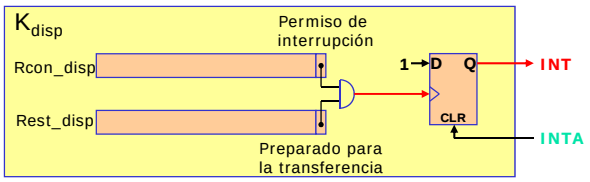
Sincronización por interrupciones

- Fases del mecanismo de sincronización por interrupciones
  6. Identificar dispositivo
  - Cada dispositivo debe ser atendido por una RAI específica.
  - La RSG ha de identificar qué controlador ha realizado la petición para saltar a ejecutar la RSI específica.
  - Se utiliza la instrucción "GETIID":
    - Envía un INTA a todos los controladores (mecanismo DAISY-CHAIN)
    - Recibe un valor "Id" que identifica el dispositivo
    - La dirección de la RSI está en "interrupts_vectors[Id]"
    - Este mecanismo recibe el nombre de interrupciones vectorizadas



Sincronización por interrupciones

- Fases del mecanismo de sincronización por interrupciones
  - 7. Eliminar la petición de interrupción
  - La petición de interrupción se mantiene hasta que es atendida. Una vez es atendida, se ha de notificar al controlador para que la desactive (INTA).
    - Software: al leer el dato correspondiente (teclado), al ejecutar el GETIID en respuesta a la INT.
    - Hardware: el procesador genera de forma automática la INTA cuando acepta la interrupción.



Sincronización por interrupciones

- Fases del mecanismo de sincronización por interrupciones

  8. Ejecutar la RSI específica
     - Una vez identificada la RSI, se ejecuta como cualquier subrutina.
     - Se vuelve a la RSG con un “JMP R6”

  9. Restaurar estado
     - Una vez de vuelta en la RSG, hay que restaurar los registros que se han salvado previamente.

  10. Retornar al programa interrumpido
     - Para retornar la programa interrumpido es necesario utilizar una instrucción especial: “RETI”.
       - Restaura el PC y la palabra de estado.

Dispositivos en SISA-F

- Reloj (identificador=0)
  - Genera una interrupción cada 0,1 segundos
  - Permiso de interrupciones: bit 0 de Rcon_rel
  - Eliminación de la petición: INTA generado por GETIID

- Teclado (identificador=1)
  - Permiso de interrupciones: bit 0 de Rcon_tec
  - Eliminación de la petición: leyendo Rdat_tec
  - Genera una interrupción al pulsar una tecla (Rest_tec0 pasa de 0 a 1)

- Disco (identificador=5)
  - Permiso de interrupciones: bit 0 de Rcon_disc
  - Eliminación de la petición: INTA generado por GETIID
  - Genera una interrupción al acabar de leer/escribir un sector (Rest_disc0 pasa de 0 a 1)

- Impresora (identificador=7)
  - Permiso de interrupciones: bit 0 de Rcon_imp
  - Eliminación de la petición: INTA generado por GETIID
  - Genera una interrupción cuando acaba de imprimir un carácter (Rest_imp0 pasa de 0 a 1)

Ejemplo E/S

- Programa que espera 10 segundos y acaba

```
int ticks;
int final;

void interrupt reloj()
{
  ticks++;
  if (ticks == 100)
    final=1;
}

main()
{
  ticks=0;
  final=0;
  di();
  interrupts_vector[0] = (void interrupt (*)()) reloj;
  out(Rcon_rel,in(Rcon_rel)| 1);
  ei();
  while (!final) ;
}
```


Ejemplo E/S

- Programa que imprime una frase acabada por "\ n"

```
char frase[30]="Esto es una frase.\n";
int final = 0;
int n = 0;
void interrupt impresora()
{
  if (frase[n]=="\n")
    final=1;
  else {
    n++;
    out(Rdat_imp, frase[n]);
    out(Rcon_imp, in(Rcon_imp) | 0x8001);
  }
}
main()
{
  di();
  interrupts_vector[7] = (void interrupt (*)()) impresora;
  out(Rdat_imp, frase[0]);
  out(Rcon_imp, in(Rcon_imp) | 0x8001);
  ei();
  while (!final) ;
}
```


Transferencia de la Información

- La pregunta que queremos resolver es:
  ¿Quién realiza la transferencia de información en una operación de E/S?
- En los ejemplos vistos, siempre era el procesador el encargado de mover la información (E/S programada):
  - Memoria → Controlador: out(Rdat_imp, c)
  - Memoria ← Controlador: c = in(Rest_imp)
- Esta situación es perfectamente válida cuando trabajamos con dispositivos que funcionan dato a dato o a frecuencias muy bajas.
- Sin embargo, ¿qué ocurre cuando hemos de transferir bloques de datos a gran velocidad (p.e. un disco para transferir una página de MV)?

Transferencia de la Información

- Ejemplo, lectura de un sector de disco:
  - El procesador programará el controlador.
  - El controlador de disco provoca una interrupción cada vez que tiene un nuevo dato.

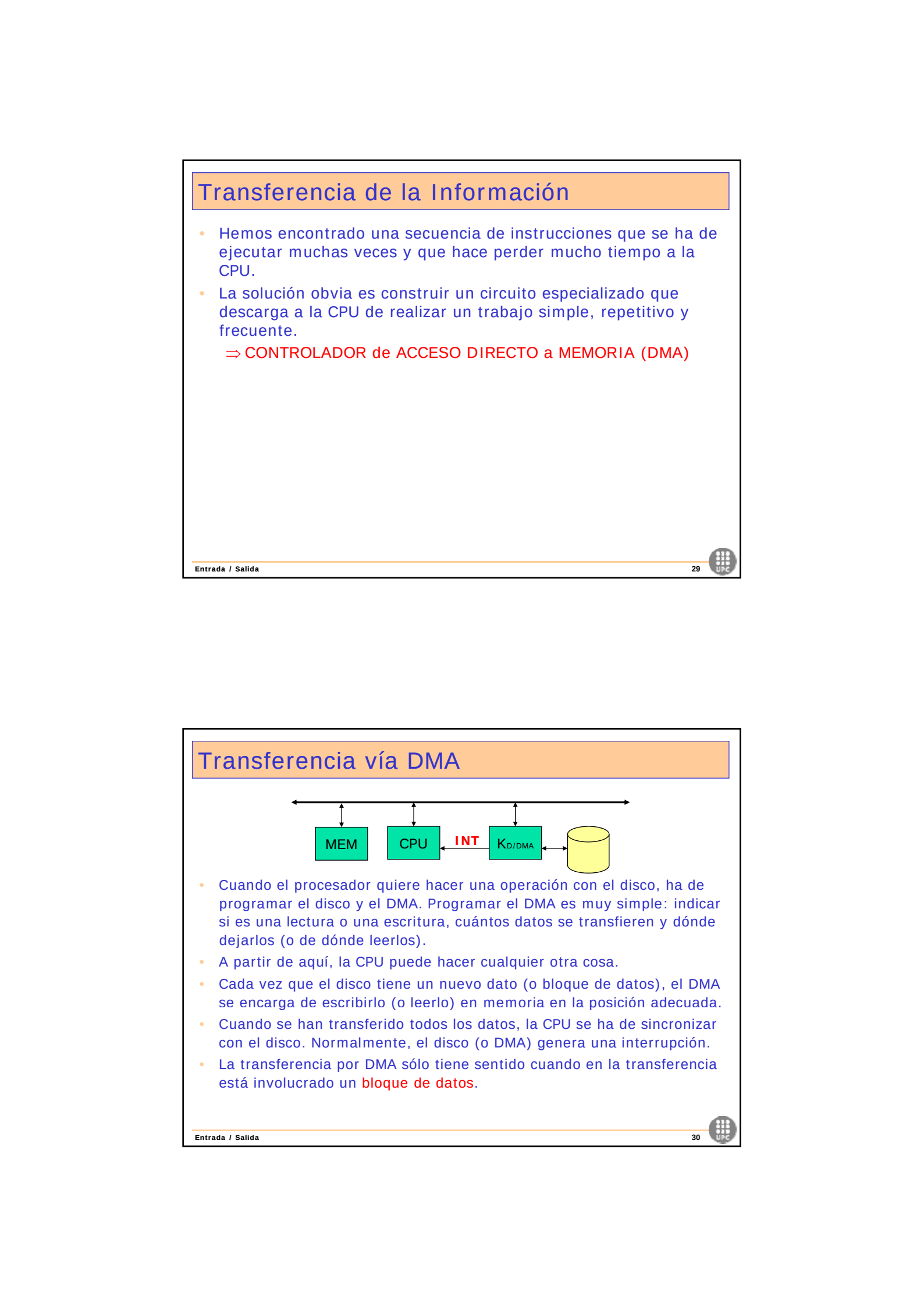
```
void interrupt disco()
{
  c = in(Rdat_dis);
  Memoria[cont]=c;
  cont++;
  if (cont == TAMSEC)
    "acabar";
}
```

Esta RAI se ejecutaría tantas veces como datos fuera a leer. Si el dispositivo es muy rápido, los datos estarán disponibles muy rápidamente, y la CPU será interrumpida constantemente y apenas podrá hacer otra cosa.

Transferencia de la Información

- Hemos encontrado una secuencia de instrucciones que se ha de ejecutar muchas veces y que hace perder mucho tiempo a la CPU.
- La solución obvia es construir un circuito especializado que descarga a la CPU de realizar un trabajo simple, repetitivo y frecuente.

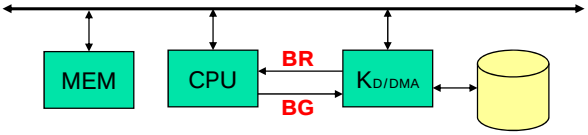
  ⇒ CONTROLADOR de ACCESO DIRECTO a MEMORIA (DMA)



Transferencia vía DMA

- Cuando el procesador quiere hacer una operación con el disco, ha de programar el disco y el DMA. Programar el DMA es muy simple: indicar si es una lectura o una escritura, cuántos datos se transfieren y dónde dejarlos (o de dónde leerlos).
- A partir de aquí, la CPU puede hacer cualquier otra cosa.
- Cada vez que el disco tiene un nuevo dato (o bloque de datos), el DMA se encarga de escribirlo (o leerlo) en memoria en la posición adecuada.
- Cuando se han transferido todos los datos, la CPU se ha de sincronizar con el disco. Normalmente, el disco (o DMA) genera una interrupción.
- La transferencia por DMA sólo tiene sentido cuando en la transferencia está involucrado un bloque de datos.

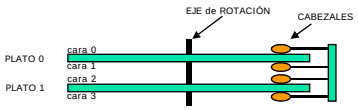
Transferencia vía DMA

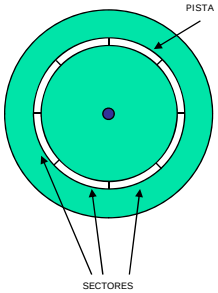


- El procesador y el KDMA se han de coordinar para acceder a Memoria sin conflictos:
  - Protocolo por robo de ciclo:
    - El procesador tiene prioridad en el acceso al bus de Memoria.
    - Cuando el KDMA necesita acceder a Memoria, activa BR.
    - Si el procesador no necesita acceder a Memoria, contesta activando BG.
    - El KDMA realiza la transferencia (1 dato).
    - Para acabar, el KDMA desactiva BR y el procesador desactiva BG.
    - El procesador tiene nuevamente acceso al bus de memoria.
  - Protocolo por transferencia a ráfaga:
    - El protocolo es similar, pero ahora en vez de transferir datos individuales, se transfieren múltiples datos antes de desactivar BG.

Dispositivos en SISA-F

- Disco (identificador=5)
  - 4 caras, 8 pistas, 16 sectores, 1024 bytes por sector.
  - Se transfieren bloques (cara-pista-sector)
  - Raddr_dis: dirección de memoria.
  - Rcara_dis, Rpist_dis, Rsect_dis.
  - Rest_disc: bit0, disco preparado; bit 15, error.
  - Rcon_disc: bit0, permiso interrupción; bit8, lectura/escritura; bit15, puesta en marcha
  - Eliminación de la petición: INTA generado por GETIID
  - Genera una interrupción al acabar de leer/escribir un sector (Rest_disc0 pasa de 0 a 1)





Ejemplo E/S

- Rutina para lanzar la lectura de un bloque de disco.

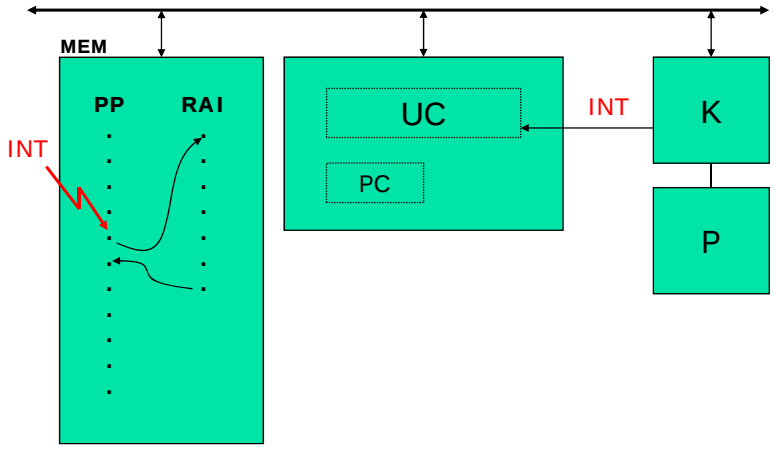
```
void LeerBloqueDMA(int cara, int pista, int sector, char *b)
{
  out(Raddr_dis, b);
  out(Rcara_dis, cara);
  out(Rpist_dis, pista);
  out(Rsect_dis, sector);
  out(Rcon_disc, (in(Rcon_disc) | 0x8001) & 0xfeff);
}
```


Ejemplo E/S

- Leer todos los sectores de la pista 3, cara 2.

```
int sector = 0;
char buffer[1024*16];
int fin = 0;
void interrupt disco()
{
  if (in(Rest_disc) == 1) { //Transferencia correcta
    if (sector==15) fin = 1;
    else { sector++;
           LeerBloqueDMA(2, 3, sector, &buffer[sector*1024]);
    }
  }
  else // ERROR: CÓDIGO GESTIÓN ERROR
}
main()
{ di();
  interrupts_vector[5] = (void interrupt (*)()) disco;
  ei();
  LeerBloqueDMA(2, 3, sector, &buffer[sector*1024]);
  while (!fin) ;
}
```


Sincronización por Interrupciones



Sincronización por Interrupciones

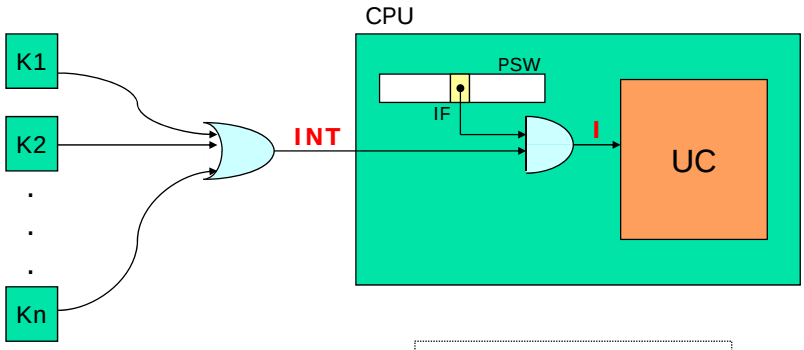
- Cualquier procesador dispone de 1 o varias entradas, a través de las cuales dispositivos externos pueden interrumpir a la CPU.
- Cuando la CPU (la UC) detecta la activación de alguna de estas señales, decide si la interrupción se sirve o no.
- En caso afirmativo, el programa en ejecución es suspendido y pasa a ejecutarse una rutina (RAI) que se encargará de tratar esa interrupción.
- Cuando la RAI acaba, el programa interrumpido reanuda su ejecución.
- Este mecanismo es transparente al programa interrumpido.
- Una RAI se comporta como una subrutina que no es activada por programa, sino por un suceso externo.

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  1. Detección de la petición de interrupción.
  2. Salvar el estado del programa interrumpido.
  3. Identificación de la RAI (en función del dispositivo que genera la interrupción).
  4. Ejecución de la rutina de atención a la interrupción (RAI).
  5. Retorno al programa interrumpido.

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  1. Detección de la petición de interrupción

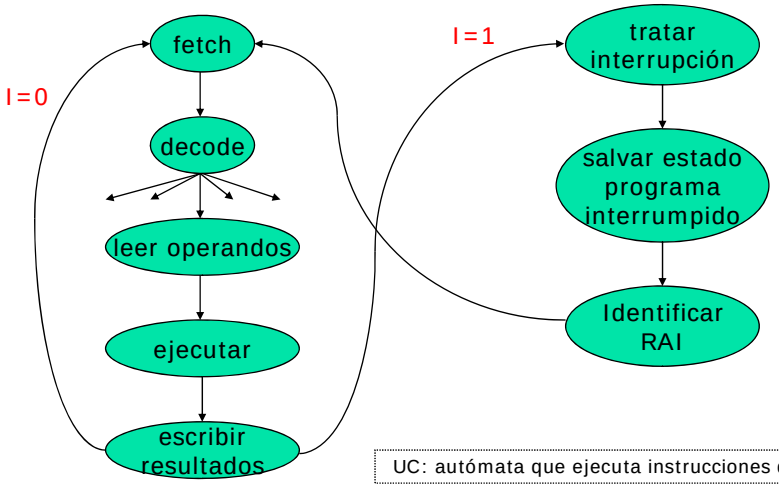


STI: IF<-1 // enable();

CLI: IF<-0 // disable();

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  1. Detección de la petición de interrupción



UC: autómata que ejecuta instrucciones de LM.

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  2. Salvar el estado del programa interrumpido
  - Un programa puede ser interrumpido en cualquier punto de su ejecución.
  - Al acabar la RAI, el programa interrumpido ha de continuar como si nunca hubiera sido interrumpido. Se han de salvar (en la pila):
    - La dirección de la siguiente instrucción a ejecutar y
    - La palabra de estado (contiene los flags)

```
cmpl %eax, %ebx
jl loop
```

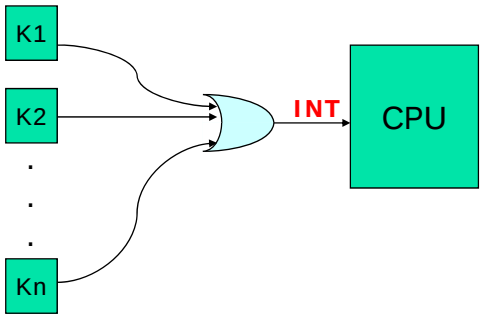
  - Es imprescindible que la RAI salve, en la pila, todos los registros que utilice.
  - Puede haber varias peticiones de interrupción en curso.

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  3. Identificación de la RAI
     - Puede haber diversos dispositivos que pueden interrumpir al procesador. Cada dispositivo necesita ser atendido por una RAI independiente.
     - Hay que identificar qué dispositivo realiza la petición de interrupción para poder seleccionar la RAI adecuada.
     - Este problema tiene 2 componentes:
       - ¿Quién ha realizado la petición de interrupción?
         → Identificación hardware / software
       - Si solicitan varios dispositivos a la vez, ¿a quién se atiende en primer lugar?
         → Hay que establecer un mecanismo de prioridades.

Sincronización por Interrupciones

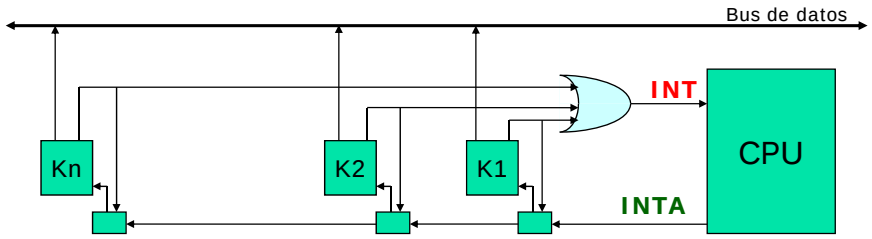
- Fases en la ejecución de una interrupción:
  3. Identificación de la RAI
     - Identificación Software



Hay una única RAI. Lo primero que hace es averiguar quién ha interrumpido (mirando los registros de estado) y actuar en consecuencia. El orden en que se miran los controladores establece la prioridad.

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  3. Identificación de la RAI
     - Identificación Hardware (mecanismo daisy chain)



El procesador responde a la petición de **INT** activando la señal de reconocimiento a la interrupción: **INTA**.
El controlador recibe la INTA y vuelca sobre el bus de datos información que permite identificar la RAI (en x86, un índice: interrupciones vectorizadas).

Sincronización por Interrupciones

- Fases en la ejecución de una interrupción:
  4. Ejecución de la rutina de atención a la interrupción (RAI)
     - Una vez identificada la RAI, ésta pasa a ejecutarse. Las operaciones a realizar dependerán del dispositivo al que se atienda y de la operación solicitada.

  5. Retorno al programa interrumpido
     - Al acabar la RAI, hay que volver al programa interrumpido. Utilizamos una instrucción especial que recupera de la pila la dirección de retorno (como en una subrutina) y la PSW (Process status word).