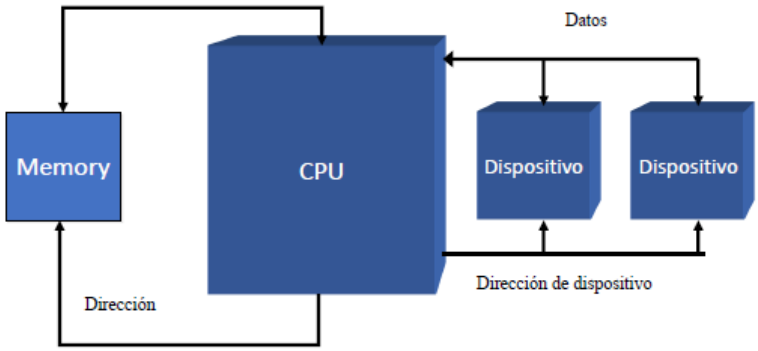
11– Programación de bajo nivel

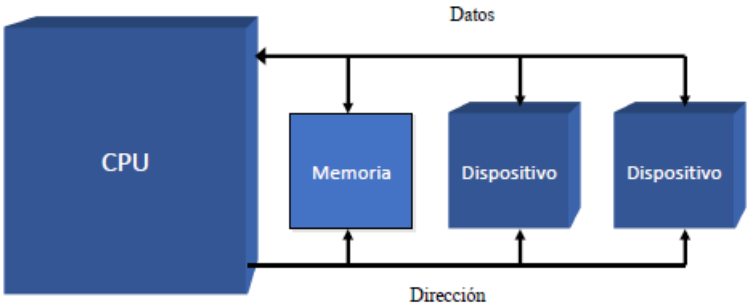
1. Mecanismos hardware de entrada/salida

Entre las principales características de un sistema embebido se encuentra la necesidad de interactuar con dispositivos de E/S de propósito específico. Un STR interactúa con su entorno a través de estos dispositivos de E/S. Los mecanismos de interacción con estos dispositivos pueden hacerse distinguiendo 2 clases generales de arquitectura:

- **Arquitectura con buses separados**: 2 buses, uno para la conexión con los dispositivos periféricos, y otro para la conexión con la memoria, de modo que ambas operaciones se podrían realizar en paralelo y sin interferencias.



- **Arquitectura sobre memoria**: se utiliza el mismo bus para conectar con los dispositivos y la memoria.



La interfaz de los dispositivos suele ser un conjunto de registros que, en función de la arquitectura, puede ser:

- **Arquitectura con buses separados**: se precisan 2 conjuntos de instrucciones en ensamblador, uno para el acceso a las posiciones de memoria, y otro para el acceso a los dispositivos. Ejemplo: iPentium e i486 usan instrucciones tipo IN AC, PORT para acceder o enviar información al dispositivo y OUT AC, PORT para recibir información del mismo.
- **Arquitectura sobre memoria**: se tiene un único mapa de memoria y, en función de la dirección, se accederá a la memoria física o a un dispositivo. Ejemplo: Motorola 68000.

Los mecanismos para controlar e interactuar con los dispositivos E/S son básicamente dos:

1. Dirigido por estatus.
2. Dirigido por interrupción.

Mecanismo de E/S dirigido por estado o estatus

Cada programa realiza comprobaciones para conocer el estatus de cada dispositivo con el que tenga que comunicarse. Una vez se conoce su estatus, el programa será capaz de realizar las acciones adecuadas y soportadas. Las clases de instrucciones hardware para este mecanismo:

- **Operaciones de test:** permiten conocer el estatus de un dispositivo.

- **Operaciones de control:** sirven para dar instrucciones al dispositivo sobre tareas u operaciones que debe realizar que no están relacionadas con transferencia de datos (Ejemplo: posicionar las cabezas de un disco).
- **Operaciones de E/S:** encargadas directamente de transferir datos entre el dispositivo y el procesador del sistema.

Hoy en día, la mayoría de los dispositivos funcionan con mecanismos dirigidos por interrupción, pero, en ocasiones, sigue siendo recomendable utilizar un mecanismo que no esté dirigido por interrupciones, ya que aumentan el grado de no determinismo de los sistemas y, si no consiguen acotarse, pueden resultar peligrosas para un sistema de tiempo real.

Mecanismo de E/S dirigido por interrupción

Si el mecanismo es dirigido por interrupción, las operaciones de E/S se efectúan de forma asíncrona a la propia ejecución del proceso, de forma paralela. Sin embargo, las interrupciones aumentan el grado de no-determinismo y entorpecen el normal funcionamiento de un STR. Así, en algunos sistemas no están permitidos o tienen una prioridad muy baja.

Una interrupción es un evento externo al que debe responder el sistema en un tiempo especificado. Las interrupciones permiten que la E/S se efectúe de manera asíncrona, lo que evita la llamada espera ocupada, que significaría tener que realizar una comprobación constante del estatus del dispositivo.

Hay tres tipos de mecanismos dirigidos por interrupción para interactuar con los dispositivos de E/S:

- **Controlado por programa**: cuando ocurre una interrupción, se suspende el proceso en ejecución para cargar un programa encargado de manejarla. Este programa efectúa las acciones necesarias y, cuando finaliza, se recupera el estado del procesador que había antes, retomando la ejecución del proceso.
- **Iniciado por programa (DMA - Acceso directo a memoria)**: cuando ocurre una interrupción se invoca a un elemento o dispositivo, denominado DMA (Direct Memory Access), el cual se encarga de gestionar esa interrupción. Este se encarga de ir realizando toda la transferencia de datos con el dispositivo y, cuando finaliza, se envía una interrupción indicando que ha finalizado, de modo que el proceso puede continuar normalmente. DMA es utilizado normalmente en aquellos dispositivos de E/S de transferencia de datos, como los discos duros. Precisamente porque el programa inicia la comunicación de E/S, pero es el dispositivo DMA el que realmente controla la transferencia de datos.
- **Controlado por programa de canal**: es una extensión del iniciado por programa. Se compone de un canal hardware, los dispositivos conectados y el programa de canal e instrucciones. No se necesita que intervenga el procesador en ningún momento, sino que directamente ese programa canal es el que se encarga de realizar la transferencia.

Los 2 últimos tipos pueden provocar robos de ciclo de acceso a memoria al procesador, y producir situaciones no deterministas. Esto significa que, mientras se están ejecutando en paralelo a la ejecución normal de otro proceso en el procesador, pueden necesitar acceder a memoria y, por tanto, roben el ciclo de acceso a memoria al proceso que se está ejecutando en el procesador.

Los elementos necesarios para los dispositivos dirigidos por interrupción son:

- **Mecanismo de cambio de contexto**

Un proceso cuando se está ejecutando en el procesador tiene un estado que consiste en el hilo de ejecución, caracterizado por el valor del contador de programa (PC), el estatus del programa y el valor de los registros de memoria. Cuando ocurre una interrupción tiene que atenderse, lo que significa invocar el manejador de la interrupción, que tiene que ejecutarse también en el procesador, sustituyendo al proceso.

Esto es lo que se denomina **cambio de contexto**, en preservar el estado del procesador para que luego pueda continuar con la ejecución del proceso donde se quedó, establecer el estado adecuado del procesador, cargando el proceso del manejador para atender a la interrupción, y la restauración del estado del procesador, para poder continuar con la ejecución del proceso que había sido suspendido (por ello se guarda el PC).

Se puede realizar a varios niveles:

- **Básico:** únicamente se guarda y actualiza el contador de programa.
- **Parcial:** guarda el contador de programa y los registros de estatus.
- **Completo:** se guarda el contexto completo del proceso y se aloja uno nuevo.

- **Mecanismos de identificación del dispositivo de la interrupción**

Estos mecanismos pueden ser:

- **Vectorizados:** se identifica el dispositivo que provoca la interrupción mediante la consulta del vector de interrupción y analizando su correspondencia con las direcciones de los dispositivos hardware disponibles. La propia interrupción tiene una información denominada **vector de interrupción**, y ahí se puede indicar el dispositivo que la ha provocado.
- **Por estatus:** se utiliza cuando existen varios dispositivos conectados al mismo controlador, esta interrupción tiene una palabra de estatus asociada que especifica el dispositivo.
- **Por sondeo:** se va preguntando a cada dispositivo cual es el que ha provocado la interrupción.

Algunas arquitecturas modernas asocian una primitiva de alto nivel al servicio de la interrupción por lo que no es absolutamente necesario el mecanismo de interrupción ya que, en esa ejecución de alto nivel, se identifica el dispositivo que ha provocado la interrupción mediante la información que viene con la propia interrupción.

- **Identificación de la interrupción**

Cuando se sabe que ha habido una interrupción y el dispositivo que la ha provocado, lo siguiente es saber de qué interrupción se trata. Se puede saber comprobando la información de estatus del dispositivo. En el momento que sabemos que dispositivo es, le preguntamos su estatus y nos indica qué es lo que está pasando. También puede ser que sea la misma interrupción la que señala su causa (en ella misma viene información relativa a que está pasando).

- **Control de interrupciones**

Permite ignorar o no las interrupciones de un dispositivo E/S dependiendo de si éstas son permitidas en el sistema. La habilitación e inhabilitación de dichas interrupciones se puede realizar de distintas maneras:

- **Control de estatus por interrupción**: proporciona un conjunto de indicadores asociados a las interrupciones, de modo que el valor de ese estatus determina qué interrupciones son las que están habilitadas y cuáles no.
- **Control por mascara de interrupción**: se tiene una palabra de máscara donde cada uno de los bits está asociado a una interrupción, de modo que el valor del bit 0 o 1 indica si esta inhabilitada o habilitada.
- **Control por nivel de interrupción**: se establece un determinado nivel a cada interrupción y, en función del nivel de ejecución del procesador, la interrupción quedará habilitada o inhabilitada si su nivel es superior o no al nivel del procesador. Esto permite establecer prioridades.

- **Control de prioridad**

Se puede establecer una prioridad para atender a las interrupciones. Esa prioridad puede venir por su nivel. Pueden existir mecanismos estáticos (se establece la prioridad en un instante inicial y queda fijada durante toda la ejecución) o dinámicos (la prioridad puede ser alterada durante la ejecución del proceso) relacionados con los mecanismos de control y los niveles de prioridad del procesador.

Ejemplo: MOTOROLA 68000

3. Interfaz de dispositivos con arquitectura sobre memoria (mismo bus para conectar la memoria y el dispositivo).
4. Tres tipos de registros que hacen de interfaz con el dispositivo: registros de control, registro de estatus y registros de buffer de datos.
5. Las interrupciones permiten que los dispositivos notifiquen el procesador que deben ser servidos. Cuenta con mecanismo de identificación vectorizado.
6. Cuando ocurre una interrupción, se almacena el PC y el estatus del procesador (suele ser una palabra, PSW), se cargan el nuevo PC y PSW del vector de interrupción (viene indicada la primitiva de servicio de la interrupción). Así, la primera palabra del vector contiene la dirección de la rutina de servicio y la segunda el PSW y la prioridad de ejecución de dicha interrupción.

2. Requisitos del lenguaje

Una de las principales características de un sistema embebido es tener que interactuar con dispositivos de E/S, cada uno con sus características. La programación de dichos dispositivos ha sido tradicionalmente hecha mediante lenguaje

ensamblador. Cada vez más, ciertos lenguajes de medio y alto nivel proporcionan mecanismos para realizar este trabajo de programación de bajo nivel.

El uso de estos lenguajes de alto nivel facilita la lectura, escritura y mantenimiento de las rutinas de control de dispositivos e interrupciones, así como el mantenimiento posterior del software desarrollado. Sin embargo, hay dos características de un lenguaje de alto nivel que es preciso proporcionar para permitir la programación de manejadores de dispositivos:

7. Mecanismos de encapsulamiento y modularidad (habituales en lenguajes como Java).
8. Modelos abstractos de manejo de dispositivos.

Mecanismos de encapsulamiento y modularidad

Las interfaces de bajo nivel de los dispositivos son dependientes de la máquina (no portables). Esto supone que es importante separar las secciones de código que son portables de las que no y encapsular, siempre que sea posible, todo el código que depende de la máquina en unidades fácilmente identificables.

En el caso de Java, tanto las clases como los paquetes son mecanismos adecuados de encapsulamiento que pueden utilizarse para este fin. Un *package* en Java es un contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes. En ADA, se emplean también mecanismos de encapsulamiento en paquetes y en C se emplea un archivo.

Modelos abstractos de manejo de dispositivos

Un dispositivo de E/S puede verse como un procesador o elemento de proceso, separado del procesador central, que efectúa cierta tarea. Se puede considerar que el sistema informático está compuesto por varios elementos de proceso que se ejecutan de manera paralela a los procesos habituales que se ejecutan en el procesador del sistema.

Estos supuestos procesos de los dispositivos deben comunicarse y sincronizarse con aquellos que se ejecutan en el procesador del sistema mediante el uso de interrupciones. Los modelos deben cubrir los siguientes aspectos:

9. Proveer mecanismos para la representación del direccionamiento y la manipulación de los registros de los dispositivos. Los dispositivos de E/S se van a contemplar como una colección de registros, por lo que hay que tener mecanismos para acceder a ellos. Según el lenguaje, cada registro puede representarse como una variable u objeto dentro del propio código del lenguaje.
10. Cubrir la representación adecuada de las interrupciones, puesto que van a ser el mecanismo de interacción normal con estos dispositivos. Esta representación puede venir dada por:
    - Procedimental: ejecución de un procedimiento. Suele ser el método más común.
    - Ejecución esporádica de un proceso (parecido al procedimental).
    - Notificación asíncrona.
    - Sincronización por variable de condición compartida.
    - Sincronización basada en mensajes: las herramientas de sincronización del SO juegan un papel fundamental en esta representación de las interrupciones (no pueden producirse interbloqueos, tampoco accesos no permitidos a secciones críticas, etc.).
11. El uso e implementación de estas opciones depende del lenguaje de programación.

3. Planificación de controladores de dispositivos

Se considera a los controladores dirigidos por programa, ya que las estrategias DMA y de programa de control de canal pueden provocar situaciones de no determinismo. A la hora de ejecutar este programa, el manejador suele tener mayor prioridad que la ejecución de un proceso ordinario para que se pueda atender la interrupción y que no se vayan acumulando. Sin embargo, no necesariamente tiene que ser así. En STR críticos puede ocurrir que algunas interrupciones tengan menos prioridad que el proceso que se esté ejecutando en un momento dado.

Considerando el análisis de tiempos con prioridad fija, en el caso de la sincronización por interrupción, cuando una interrupción libera una tarea esporádica para la ejecución, se tiene que considerar el coste del manejador de interrupciones en sí, ya que puede darse que la prioridad del controlador sea mayor que la de la tarea, con lo que tareas con una prioridad mayor, pero menor que el manejador, se verían interferidas. Así, se podría producir lo que se denomina una **inversión de la prioridad**.

El protocolo a utilizar para emplear un dispositivo dirigido por estatus es el siguiente:

1. Realizar una petición de lectura.
2. Esperar hasta que el hardware efectúe esa lectura. Esta espera tiene como hándicap que el tiempo durante el cual está esperando, hay que gestionarlo. Existen 3 estrategias:
    - Espera ocupada: el sistema espera hasta que el hardware termine de efectuar la lectura. Esto solamente es viable o razonable en los casos donde la lectura ocupe poco tiempo (si no, estaría el sistema parado esperando a que el sistema periférico terminara de hacer su función sin realizar nada más).
    - Replanificar el proceso para un instante posterior entendiendo por un instante posterior aquel en el que la operación de lectura o escritura ya haya finalizado. Por ejemplo, enviando un bloque de información a un dispositivo de almacenamiento, el proceso que atiende esa interrupción manda el comando o la orden, y lo replanifica para un instante posterior, en el que se presupone que ya habrá terminado de realizar esa operación.
    - Para procesos periódicos, consiste en la separación de la acción entre periodos: si el proceso se ejecuta periódicamente aprovechamos esos periodos para enviar el siguiente comando o atender la siguiente interrupción y, al mismo tiempo, leer la información disponible proveniente de la interrupción anterior.
3. Una vez que el hardware ha efectuado la lectura, y la ha depositado en los registros, hay que acceder a los mismos para leer esa información desde el programa.

4. Gestión de la memoria

Los STR suelen tener limitada la cantidad de memoria disponible (un sistema empotrado suele tener un tamaño acorde a la función que tiene que cumplir y por tanto no caben sistemas de almacenamiento más allá de lo estrictamente necesario) por lo que es necesario garantizar una utilización eficiente de esta memoria.

En esta gestión eficiente hay que señalar 2 estructuras de datos presentes en el modelo de gestión de memoria de la mayoría de los lenguajes de programación:

- Montón (*heap*).
- Pila (*stack*).

Gestión del montón (*heap*)

Un **montón** (***heap***) es una región de memoria disponible para las solicitudes de memoria dinámica por parte del SO. Es decir, cuando se necesita reservar memoria mediante una operación new para un objeto, esa memoria reservada se ubica en una zona denominada *heap*. Estas reservas de memoria se deben realizar de una forma eficiente y es necesario, por tanto, gestionar las reservas y liberaciones para hacerlo de forma óptima.

En cuanto a la reserva de memoria, suele estar implícito en el propio código de programación (cuando se declara una variable, el compilador reserva memoria para ella). En cuanto a la liberación, suele haber más problema, por lo que hay varias estrategias:

- El programador devuelve la memoria explícitamente empleando instrucciones de liberación cuando ya no la vaya a utilizar. Por ejemplo, el lenguaje C.
- Monitorizar por parte del sistema la memoria y determinar cuándo no se va a acceder a ella para liberarla. En este caso, por ejemplo, podrían ser aquellas variables o aquellas secciones de memoria para las cuales ya no se tiene ningún puntero ni nada que apunte a ellas y, por tanto, ya no son accesibles.
- Monitorizando por parte del sistema la memoria y determinando cuándo no va a ser usada. No sólo cuando no sea accesible sino cuando no va a ser usada. En este caso hacen falta estrategias de predicción de qué memoria, qué variables o qué objetos no van a ser usados más.

Este sistema de predicción y esta monitorización de memoria lo realiza un proceso denominado **recolector de basura**, una forma de liberar la memoria donde el sistema monitoriza la memoria y libera los fragmentos que ya no se utilizan. La recolección de basura se puede realizar, bien cuando la memoria montón está llena, o bien incrementalmente.

No obstante, la ejecución de esta recolección de basura puede tener un impacto sobre la predictibilidad en la ejecución de los procesos, puesto que no es posible determinar cuándo va a entrar en juego y cuándo va a provocar retardos en el sistema porque esté liberando memoria.

En RT-Java, todos los objetos se reservan en el montón, por lo que será necesario un recolector de basura para su correcta implementación. Pero como esto puede producir no determinismos, RT-Java presenta áreas de memoria para evitar este comportamiento. Estos tipos de memoria pueden ser de dos clases:

- **Memoria inmortal:** es un área de memoria compartida por todos los hilos de la aplicación y no padecen el recolector de basura. Se libera únicamente cuando termina el programa.
- **Memoria con alcance:** se reserva espacio para objetos con tiempo de vida bien definido. A cada memoria con alcance se le asocia un contador de referencias, indicando cuántos procesos tienen acceso a la misma y, cuando este contador es 0, se libera la memoria del objeto asociada.

Gestión de la pila (*stack*)

La **pila** es un aspecto fundamental para la correcta ejecución de código, y su tamaño es un aspecto a considerar. Hay que cuidar el tamaño de la memoria para utilizar únicamente aquella que sea necesaria. Para determinar este tamaño es necesario conocer el comportamiento de ejecución de cada tarea para poder predecir así sus necesidades de memoria de pila. Para ello, es útil utilizar herramientas para el análisis del flujo de control a partir del código de la tarea y así poder estimar cuanta memoria será la que finalmente necesiten.