Practica 4: Manejo de Entradas y Salidas

Rolando Suarez V- 30445947
Jesus Muñoz V- 27188137

1 Explicación del Ciclo de una Interrupción de Hardware

El ciclo de una interrupción de hardware es un proceso que permite que los dispositivos periféricos (como el teclado, el disco duro, la tarjeta de red, etc.) se comuniquen con el procesador para solicitar atención.

1. **Ocurrencia del evento** El ciclo de la interrupción comienza cuando un dispositivo periférico necesita la atención del procesador. Por ejemplo:
   - Un usuario presiona una tecla en el teclado.
   - El disco duro ha terminado de leer un bloque de datos y lo tiene listo para el procesador.
   - La tarjeta de red ha recibido un paquete de datos.
   - Un temporizador programado llega a cero.
2. **Generación de la señal de interrupción** Una vez que el evento ocurre, el dispositivo periférico genera una señal eléctrica en una línea de hardware dedicada, conocida como línea de interrupción (IRQ, *Interrupt Request Line*). Esta señal viaja a través del bus del sistema hasta el controlador de interrupciones programable (PIC, *Programmable Interrupt Controller*), o un componente similar en arquitecturas más modernas.
3. **Recepción de la interrupción por el PIC** El PIC es un chip de hardware diseñado para gestionar múltiples solicitudes de interrupción de diferentes dispositivos. El PIC recibe la señal de interrupción del dispositivo periférico y la prioriza. Si la interrupción es de mayor prioridad que la que se está ejecutando actualmente (o si no se está ejecutando ninguna), el PIC:
   - Envía una señal de interrupción al procesador a través de su pin de interrupción (conocido como pin `INTR`).
   - En algunas arquitecturas, el procesador puede estar ejecutando una interrupción de mayor prioridad o puede tener las interrupciones deshabilitadas. En este caso, el PIC mantiene la interrupción pendiente hasta que el procesador esté disponible.
4. **El procesador recibe la interrupción** El procesador monitorea su pin `INTR` en cada ciclo de instrucción. Cuando el procesador detecta una señal en este pin, detiene su ejecución actual al finalizar la instrucción en curso. Antes de atender la interrupción, el procesador realiza los siguientes pasos:
   - **Guarda el estado del procesador:** El procesador guarda el estado actual del programa que se está ejecutando. Esto incluye el valor del contador de programa (PC, *Program Counter*), los registros de la CPU, y otros datos importantes en un área especial de la memoria conocida como la pila (*stack*). Esto es crucial para poder reanudar el programa original desde donde se quedó.
   - **Envía un reconocimiento al PIC:** El procesador envía una señal de reconocimiento al PIC para indicar que está listo para atender la interrupción.
5. **El PIC envía el vector de interrupción** Al recibir el reconocimiento del procesador, el PIC responde enviando un número de interrupción (o *vector de interrupción*) que identifica de manera única al dispositivo que generó la interrupción. Este número es un índice a una tabla en la memoria, conocida como la Tabla de Vectores de Interrupción (IVT, *Interrupt Vector Table*).

6. **El procesador busca el manejador de interrupción** El procesador usa el vector de interrupción recibido para buscar en la Tabla de Vectores de Interrupción la dirección de memoria del Manejador de Interrupción (o ISR, *Interrupt Service Routine*) correspondiente. Cada dispositivo periférico tiene su propio ISR, que es un pequeño programa diseñado para atender específicamente la solicitud de ese dispositivo.

7. **Ejecución del Manejador de Interrupción (ISR)** El procesador salta a la dirección de memoria del ISR y comienza a ejecutar sus instrucciones. El ISR se encarga de:

   - Realizar la tarea solicitada por el dispositivo (por ejemplo, leer la tecla presionada del buffer del teclado, transferir los datos del disco duro a la memoria RAM, etc.).
   - Enviar un comando al dispositivo periférico para que reinicie su estado o indique que la solicitud ha sido atendida.
   - Enviar un comando de fin de interrupción (EOI, *End-of-Interrupt*) al PIC.

8. **Fin del ciclo** Una vez que el ISR termina su ejecución, se restaura el estado del procesador que se había guardado en la pila. Esto incluye cargar de nuevo los valores de los registros y el contador de programa. El procesador continúa la ejecución del programa que se había interrumpido, exactamente desde el punto donde se detuvo.

2 Diferencias entre gestionar E/S con sondeo y hacerlo con interrupciones

Gestionar la Entrada/Salida (E/S) es un aspecto crítico de cualquier arquitectura de computadoras. Existen dos enfoques principales para que la CPU se comunique con los dispositivos periféricos: el sondeo (*polling*) y las interrupciones.

2.1 E/S por sondeo (Polling)

El sondeo es una técnica de E/S programada en la que el procesador verifica activamente el estado de un dispositivo periférico para determinar si está listo para una operación de E/S.

- **Mecanismo:** El procesador ejecuta un bucle de espera, leyendo repetidamente un registro de estado de un dispositivo periférico (mapeado en memoria, es decir, *Memory-Mapped I/O*) para comprobar una bandera o *flag* que indica si la operación de E/S ha terminado o si el dispositivo está listo para transferir datos. El procesador no puede hacer otra cosa mientras está en este bucle.

2.2 E/S por interrupciones

Las interrupciones son un mecanismo de hardware en el que un dispositivo periférico alerta al procesador cuando necesita atención, sin que la CPU tenga que estar monitoreando constantemente.

- **Mecanismo:** El dispositivo periférico genera una señal de interrupción en una línea IRQ. Esta señal hace que el procesador suspenda la ejecución del programa actual, guarde su estado y salte a una rutina especial llamada el Manejador de Interrupción (ISR, *Interrupt Service Routine*). Después de que el ISR atiende la solicitud del dispositivo, el procesador restaura su estado y reanuda el programa original.

2.3 Cuadro comparativo

Table 1: Cuadro comparativo entre Sondeo e Interrupciones

Característica	Sondeo (Polling)	Interrupciones
Mecanismo	El procesador comprueba periódicamente el estado del dispositivo.	El dispositivo avisa al procesador cuando está listo.
Flujo de control	La CPU controla el flujo de manera activa.	El dispositivo controla el flujo de manera pasiva (asíncrona).
Uso de la CPU	Ineficiente; la CPU está ocupada en un bucle de espera.	Eficiente; la CPU puede realizar otras tareas.
Latencia	Predecible (depende de la frecuencia del sondeo).	Variable (depende de si las interrupciones están habilitadas, la prioridad y la sobrecarga).
Complejidad	Baja. Fácil de implementar.	Alta. Requiere hardware y software específicos.
Casos de uso	Sistemas empotrados simples, dispositivos muy rápidos o cuando se espera que un evento ocurra con mucha frecuencia.	Sistemas operativos, multitarea, dispositivos lentos, eventos impredecibles.

3 Ventajas del uso de interrupciones en términos de uso del procesador

El uso de interrupciones para el manejo de periféricos tiene varias ventajas significativas en términos de la eficiencia y el uso del procesador, especialmente en comparación con el sondeo (*polling*):

1. **Aprovechamiento eficiente del tiempo de la CPU:** A diferencia del sondeo, donde el procesador pierde ciclos de reloj en un bucle de espera, las interrupciones permiten que el procesador se ocupe de otras tareas. El procesador solo es interrumpido y dedica tiempo a un dispositivo cuando este realmente lo necesita. Esto es fundamental para la multitarea, ya que permite que la CPU ejecute múltiples programas de manera concurrente.

2. **Respuesta inmediata a eventos:** Las interrupciones permiten que la CPU responda a eventos inesperados o asíncronos de manera casi instantánea, sin importar lo que esté haciendo en ese momento. Esto es crucial para dispositivos que generan eventos infrecuentes pero críticos, como la pulsación de una tecla o una señal de error.

3. **Reducción de la carga del procesador:** Con las interrupciones, la CPU no necesita dedicar recursos constantemente a la verificación del estado de los dispositivos. Esto libera ciclos de CPU para tareas más productivas, mejorando el rendimiento general del sistema.

4. **Escalabilidad:** Las interrupciones facilitan la gestión de un gran número de dispositivos de entrada/salida. Cada dispositivo puede tener su propia línea de interrupción y un manejador de interrupción (ISR) específico, lo que permite al sistema gestionar de forma eficiente una variedad de periféricos sin que el procesador se vea abrumado por la necesidad de sondear cada uno de ellos.

4 Registros especiales para gestionar interrupciones en MIPS32

En la arquitectura MIPS32, la gestión de interrupciones y excepciones se realiza a través de un conjunto de registros especiales que forman parte del Coprocesador 0 (CP0). El CP0 se utiliza para tareas de control del sistema, como la gestión de memoria y el manejo de excepciones.

1. **Registro `Status` (CP0, registro 12):**

- Este registro controla el estado del procesador y las interrupciones.
- Contiene un bit de habilitación global de interrupciones (`IE`), que si está a 0, inhabilita todas las interrupciones.
- También contiene un conjunto de máscaras de interrupción (`IM`), donde cada bit corresponde a una línea IRQ específica. Si un bit de la máscara está a 0, la interrupción asociada a esa línea está deshabilitada, aunque el bit de habilitación global esté a 1.
- Otros bits controlan el modo del procesador (kernel o usuario) y el nivel de excepción.

2. **Registro `Cause` (CP0, registro 13):**
   - Este registro almacena la razón por la que se produjo la interrupción o excepción.
   - Contiene un campo llamado código de excepción (`ExcCode`) que indica la causa de la interrupción, por ejemplo, si fue una interrupción de hardware, una excepción de alineación de memoria, una división por cero, etc.
   - Incluye un conjunto de bits de interrupción pendiente (`IP`) que reflejan el estado de las líneas IRQ. Si un dispositivo activa su línea IRQ, el bit correspondiente en este registro se establece. El manejador de interrupciones lee estos bits para determinar qué dispositivo necesita ser atendido.
3. **Registro `EPC` (*Exception Program Counter*, CP0, registro 14):**
   - Cuando ocurre una interrupción o excepción, el hardware del MIPS salva automáticamente la dirección de la instrucción en la que se produjo el evento en el registro `EPC`.
   - El manejador de interrupciones utiliza este registro para saber dónde debe reanudar la ejecución del programa una vez que ha terminado de atender la interrupción. La instrucción `eret` (*exception return*) carga el valor de `EPC` en el contador de programa (PC) para regresar al punto de interrupción.

Para interactuar con estos registros, el procesador MIPS utiliza instrucciones especiales como `mfc0` (*move from coprocessor 0*) y `mtc0` (*move to coprocessor 0*), que permiten a la CPU leer y escribir datos en los registros del Coprocesador 0.

5 Necesidad de guardar el contexto al entrar en una rutina de servicio

Es crucial guardar el contexto del procesador (los registros) al entrar en una rutina de servicio de interrupción (ISR, *Interrupt Service Routine*) por la razón de garantizar que el programa que se interrumpió pueda reanudarse correctamente y sin errores.

1. **La rutina de interrupción es un programa más:** La ISR es, en esencia, una pieza de código que, al igual que cualquier otro programa, necesita usar los registros de la CPU para realizar sus operaciones (almacenar variables temporales, realizar cálculos, etc.).
2. **Sobreescritura de registros:** Si la ISR utilizara los registros de la CPU sin guardarlos previamente, sobrescribiría los valores que el programa principal estaba utilizando en el momento de la interrupción. Por ejemplo, si el programa estaba usando el registro `$t0` para almacenar una variable importante, y la ISR también usa `$t0`, el valor original se perdería para siempre.
3. **Restauración del estado para una reanudación transparente:** Al finalizar la ISR, el procesador debe regresar al programa original y continuar ejecutándose desde el punto exacto donde se detuvo. Para que esto sea posible, necesita que el estado de la CPU sea exactamente el mismo que antes de la interrupción. Esto incluye el valor de todos los registros. Si los valores de los registros no se restauran, el programa original operaría con datos incorrectos, lo que probablemente causaría un fallo, un resultado erróneo o un comportamiento inesperado.

6 Momentos en que pueden generarse excepciones en un sistema MIPS32

6.1 a) Enumeración de situaciones

A continuación se enumeran al menos cuatro situaciones en las que se puede generar una excepción:

1. **Excepción de Dirección (*Address Error*):** Se produce cuando el procesador intenta acceder a una dirección de memoria que no está correctamente alineada. Por ejemplo, al intentar cargar una palabra (`lw`) desde una dirección que no es un múltiplo de 4, o al intentar ejecutar una instrucción desde una dirección que no es un múltiplo de 4.

2. **Desbordamiento Aritmético (*Arithmetic Overflow*):** Ocurre cuando el resultado de una operación aritmética con signo (como `add` o `sub`) es demasiado grande o demasiado pequeño para ser representado en los 32 bits del registro destino. En MIPS, las instrucciones como `addu` y `subu` (sin signo) no causan una excepción de desbordamiento.

3. **Instrucción Ilegal (*Illegal Instruction*):** Se genera cuando el procesador encuentra un código de operación (*opcode*) en la memoria de instrucciones que no reconoce. Esto puede ocurrir debido a un error en el programa o a la corrupción de la memoria de instrucciones.

4. **Llamada al Sistema (*System Call*):** Aunque es una excepción generada intencionalmente por el programa, se maneja de la misma manera que las demás. La instrucción `syscall` permite que un programa de usuario solicite servicios del sistema operativo, como acceso a E/S, creando una excepción que transfiere el control del procesador al *kernel*.

5. **Excepción por Entrada/Salida (E/S) o Interrupción de Hardware:** Un dispositivo externo (teclado, tarjeta de red, etc.) puede generar una señal de interrupción para solicitar la atención del procesador. Aunque a menudo se les llama "interrupciones", se gestionan a través del mismo mecanismo de excepción del MIPS.

6.2 b) Etapas del *pipeline* que pueden provocar una excepción

El *pipeline* de MIPS32 estándar de cinco etapas (IF, ID, EX, MEM, WB) es susceptible a diferentes tipos de excepciones en sus distintas fases:

1. **Etapa IF (*Instruction Fetch* - Búsqueda de Instrucción):**
   - **Excepción de Dirección:** Si la dirección de la instrucción en el contador de programa (PC) no está alineada correctamente (no es un múltiplo de 4), se genera una excepción de dirección en esta etapa, antes de que la instrucción pueda ser buscada en la memoria.

2. **Etapa ID (*Instruction Decode* - Decodificación de Instrucción):**
   - **Instrucción Ilegal:** El procesador lee el *opcode* de la instrucción en esta etapa para determinar qué operación debe realizar. Si el *opcode* no coincide con ninguna de las instrucciones válidas del conjunto, se detecta una excepción de instrucción ilegal.
   - **Llamada al Sistema (`syscall`):** La instrucción `syscall` es reconocida en esta etapa, y el hardware está diseñado para generar una excepción cuando se encuentra.

3. **Etapa EX (*Execute* - Ejecución):**
   - **Desbordamiento Aritmético:** Las operaciones aritméticas se realizan en esta etapa. El hardware de la ALU (*Arithmetic Logic Unit*) detecta si el resultado de una operación con signo, como `add`, ha excedido el rango de 32 bits, lo que provoca la excepción de desbordamiento.

4. **Etapa MEM (*Memory Access* - Acceso a Memoria):**

- **Excepción de Dirección:** Las instrucciones de carga y almacenamiento (`lw`, `sw`, etc.) acceden a la memoria en esta etapa. Si la dirección de datos a la que intentan acceder no está alineada (por ejemplo, una `lw` a una dirección no múltiplo de 4), se genera una excepción de dirección.
- **Excepción de acceso a memoria:** Problemas de protección de memoria o fallos en la traducción de direcciones (en sistemas con memoria virtual, por ejemplo, un *TLB miss* o un *TLB fault*) también se detectan en esta etapa.

5. **Etapa WB (*Write Back* - Escritura de Resultado):**

   - Normalmente, esta etapa no provoca excepciones. Su función es simplemente escribir el resultado de una operación en un registro. Sin embargo, la excepción detectada en una etapa anterior (como el desbordamiento en la etapa EX) evita que la instrucción complete esta fase, y el *pipeline* debe ser vaciado para manejar la excepción.

7 Estrategias de tratamiento de excepciones e interrupciones

7.1 a) Diferencias entre interrupciones y excepciones

Aunque ambos términos a menudo se usan de forma intercambiable para referirse a la interrupción del flujo de ejecución de un programa, en la arquitectura de computadoras tienen diferencias clave, especialmente en su naturaleza temporal.

- **Interrupciones (*Interrupts*):**
  - **Naturaleza:** Son **asíncronas**. Ocurren en un momento impredecible, independiente de la instrucción que se esté ejecutando en la CPU en ese instante.
  - **Causa:** Son generadas por eventos externos al procesador, es decir, por hardware periférico (como un teclado, un disco duro, un temporizador, una tarjeta de red).
  - **Ejemplo:** Un usuario presiona una tecla, y el teclado genera una señal de interrupción para notificar a la CPU.
- **Excepciones (*Exceptions*):**
  - **Naturaleza:** Son **síncronas**. Son el resultado directo de la ejecución de una instrucción. Ocurren en un punto predecible en el flujo del programa.
  - **Causa:** Son generadas internamente por el propio procesador cuando se detecta una condición de error o un evento especial.
  - **Ejemplo:** Una instrucción de suma provoca un desbordamiento aritmético, o el programa ejecuta una instrucción de llamada al sistema (`syscall`).

7.2 b) Estrategias de tratamiento de excepciones e interrupciones en MIPS32

En MIPS32, el manejo de excepciones e interrupciones se realiza a través de un mecanismo unificado. Las estrategias principales para redirigir la ejecución hacia la rutina de servicio son:

1. **Estrategia de punto de entrada único (*Non-Vectored*):**
   - En este modelo, todas las excepciones e interrupciones, sin importar su origen, hacen que el procesador salte a la misma dirección de memoria (una dirección fija y predefinida, como `0x80000180` en el modo *kernel*).
   - La primera parte del código en esa dirección es un manejador de excepciones general (*general exception handler*).
   - Este manejador general es responsable de leer un registro especial, como el registro `Cause` (del Coprocesador 0), para determinar la causa exacta de la excepción.

- Una vez que se identifica la causa, el manejador general utiliza instrucciones de salto o bifurcación para redirigir la ejecución al manejador específico de la interrupción o excepción (ISR).
- **Ventaja:** Este método es más simple de implementar en el hardware, ya que solo necesita un punto de entrada.
- **Desventaja:** Introduce una pequeña latencia, ya que el procesador debe "sondear" el registro `Cause` antes de ejecutar el código específico.

2. **Estrategia de punto de entrada con Vectores (*Vectored Interrupts*):**
   - En arquitecturas MIPS más avanzadas o configuradas para ello, se puede utilizar una versión de esta estrategia. El procesador puede saltar a diferentes direcciones de memoria para tipos de excepciones específicos.
   - Esto se logra configurando el bit `VT` (*Vector*) en el registro `Status` y utilizando el registro `EBase` (*Exception Base*) para definir la dirección base de la tabla de vectores de excepción.
   - Cada causa de excepción tiene un *offset* que se suma a la dirección base para encontrar la dirección del manejador de servicio específico.
   - **Ventaja:** Elimina el paso de sondear el registro `Cause`, lo que reduce la latencia de manejo de excepciones.

7.3 La función del registro EPC (*Exception Program Counter*)

El registro `EPC` es crucial para la gestión de excepciones e interrupciones. Su función principal es almacenar la dirección de retorno del programa que fue interrumpido.

- Cuando ocurre una excepción, el hardware del MIPS guarda automáticamente el valor del contador de programa (PC) en el registro `EPC`.
- El valor que se guarda en `EPC` depende de la naturaleza de la excepción:
  - Para excepciones síncronas (como `syscall` o desbordamiento), `EPC` almacena la dirección de la instrucción que causó la excepción.
  - Para interrupciones asíncronas, `EPC` almacena la dirección de la instrucción que se habría ejecutado a continuación.
- Una vez que la rutina de servicio ha terminado su tarea, el procesador utiliza la instrucción especial `eret` (*exception return*). Esta instrucción lee el valor de `EPC` y lo carga de nuevo en el PC, permitiendo que el programa original continúe su ejecución exactamente desde donde se detuvo.

8 Habilitación de interrupciones en dispositivos y procesador

8.1 a) Habilitación de interrupciones en el teclado, la pantalla y el procesador

La habilitación de interrupciones es un proceso de dos pasos: primero se habilita en el dispositivo periférico y luego en el procesador.

- **En el Teclado:** Las interrupciones en un teclado se habilitan a través del controlador de teclado. El procesador, a través del sistema operativo, habilita la interrupción del teclado escribiendo un valor específico en un registro de control del controlador de teclado. Este registro está mapeado en memoria (E/S mapeada en memoria) en una dirección específica. El *driver* del teclado ejecuta una instrucción de escritura (`sw`) a esa dirección, lo que activa la lógica del controlador para que genere una señal de interrupción cada vez que una tecla es presionada o liberada.

- **En la Pantalla:** La habilitación de interrupciones en un dispositivo de pantalla (como una tarjeta gráfica) se realiza de manera similar. La pantalla puede generar interrupciones para eventos como el retraso de barrido vertical (*vertical retrace*), que es el período en el que el haz de electrones de un monitor de tubo de rayos catódicos (CRT) se mueve de la parte inferior a la superior de la pantalla. El *driver* de la pantalla habilita esta interrupción escribiendo un valor a un registro de control específico del controlador de vídeo, también accesible a través de E/S mapeada en memoria.
- **En el Procesador (MIPS32):** Para que el procesador MIPS32 responda a cualquier señal de interrupción, se deben habilitar las interrupciones a través de los bits del registro de estado (*Status*) del Coprocesador 0 (CP0, registro 12). Se requieren dos niveles de habilitación:
  1. **Habilitación global:** El bit `IE` (*Interrupt Enable*) en el registro `Status` debe ser establecido a 1. Si este bit está a 0, todas las interrupciones (sin importar la configuración de la máscara) serán ignoradas por el procesador.
  2. **Habilitación de máscaras:** Los bits `IM` (*Interrupt Mask*) del registro `Status` (bits 8-15) deben ser modificados. Estos bits corresponden a las 8 líneas de interrupción de hardware del sistema (`IP0` a `IP7`). Para habilitar una interrupción específica, el bit correspondiente en la máscara debe ser establecido a 1.

  El proceso para habilitar una interrupción específica en el procesador sería:
  - Leer el valor actual del registro `Status` usando la instrucción `mfc0`.
  - Modificar el bit `IE` y el bit de máscara (`IM`) de la interrupción deseada, estableciéndolos a 1.
  - Escribir el nuevo valor modificado de vuelta al registro `Status` usando la instrucción `mtc0`.

8.2 b) ¿Qué pasaría si habilitamos interrupciones en los dispositivos, pero no en el procesador?

Si las interrupciones están habilitadas en un dispositivo periférico (como el teclado), pero el bit de habilitación global (`IE`) en el registro `Status` del procesador MIPS32 está a 0, ocurre lo siguiente:

1. **El dispositivo genera la interrupción:** Cuando el evento ocurre (por ejemplo, se presiona una tecla), el dispositivo genera la señal de interrupción en su línea IRQ.
2. **La interrupción llega al procesador:** La señal de interrupción llega al procesador y hace que el bit de interrupción pendiente (`IP`) correspondiente en el registro `Cause` del Coprocesador 0 se establezca a 1.
3. **El procesador la ignora:** Dado que el bit `IE` en el registro `Status` está a 0, el procesador está configurado para ignorar cualquier señal de interrupción externa. El flujo de ejecución del programa que se está ejecutando actualmente no se detiene.
4. **La interrupción queda pendiente:** La señal de interrupción se mantiene en el bit `IP` del registro `Cause` del procesador, esperando a ser reconocida. El procesador continuará ejecutando su programa normal, sin saltar al manejador de interrupciones.

9 Procesamiento de interrupciones

9.1 a) Describe paso a paso qué ocurre cuando se produce una interrupción de reloj

1. **Ocurrencia del evento (Hardware):**
   - El temporizador del sistema (un chip de hardware) cuenta hasta cero o alcanza un valor predefinido.

- En ese momento, el chip genera una señal eléctrica en su línea de interrupción (IRQ).

2. **Detección y respuesta del procesador (Hardware):**
   - El procesador, si tiene las interrupciones habilitadas (`Status.IE=1` y el bit de máscara correspondiente en `Status.IM` a 1), detecta esta señal al finalizar la instrucción que está ejecutando en ese momento.
   - El hardware de la CPU realiza las siguientes acciones automáticamente:
     - Completa la instrucción actual.
     - Guarda la dirección de la siguiente instrucción a ejecutar en el registro `EPC` (*Exception Program Counter*).
     - Almacena el motivo de la interrupción (en este caso, una interrupción de hardware del reloj) en el registro `Cause`.
     - Cambia el modo de operación del procesador a modo *kernel* y deshabilita temporalmente nuevas interrupciones manipulando el registro `Status` para evitar interrupciones anidadas.
     - Salta a la dirección de memoria predefinida para la rutina de manejo de interrupciones (el vector de interrupción, por ejemplo, `0x80000180`).

3. **Ejecución de la rutina de servicio (Software):**
   - El código que se encuentra en la dirección de salto es el manejador de interrupciones del sistema operativo.
   - El software realiza las siguientes acciones:
     - **Prólogo:** Antes de hacer cualquier otra cosa, el manejador de interrupciones general guarda los valores de los registros generales de propósito del procesador en la pila del *kernel*. Esto asegura que el programa interrumpido no pierda sus datos.
     - **Identificación:** El código lee el registro `Cause` para identificar que se trata de una interrupción de reloj.
     - **Llamada al manejador específico:** El manejador general salta al manejador específico del temporizador.
     - **Servicio del temporizador:** El ISR del temporizador realiza su tarea, que puede incluir:
       * Actualizar el contador de tiempo del sistema.
       * Decrementar la cuenta de tiempo asignada al proceso actual.
       * Si el tiempo del proceso ha expirado, invocar el *scheduler* del sistema operativo para realizar un cambio de contexto a otro proceso.
     - **Reconocimiento:** El ISR escribe un valor en un registro del chip del temporizador para reconocer que la interrupción ha sido atendida. Esto prepara al temporizador para generar la próxima interrupción.
     - **Epílogo:** El manejador de interrupciones restaura los valores de los registros de propósito general desde la pila.

4. **Retorno del control (Hardware y Software):**
   - El manejador de interrupciones ejecuta la instrucción `eret` (*exception return*).
   - `eret` es una instrucción de hardware que realiza lo siguiente:
     - Restaura el valor del registro `Status` a su estado anterior, habilitando de nuevo las interrupciones y devolviendo el procesador al modo de usuario.
     - Carga el valor del registro `EPC` de vuelta al contador de programa (PC).
   - La CPU continúa la ejecución de la instrucción en la que se había quedado el programa original.

Registros que se guardan y restauran:

- `EPC` y `Cause`: Guardados automáticamente por el hardware.
- Registros de propósito general (`$t0`, `$s0`, etc.): Guardados y restaurados por el software (la rutina de servicio) en la pila.
- `Status`: Modificado automáticamente por el hardware al entrar en la rutina y restaurado por la instrucción `eret`.

9.2 b) ¿Por qué es importante guardar el contexto?

Guardar el contexto (los registros generales, el `EPC` y el `Status`) al entrar en una rutina de interrupción es fundamental para garantizar la transparencia del proceso.

- Para el programa interrumpido, la interrupción debe ser invisible. Es decir, el programa debe poder reanudar su ejecución como si nunca se hubiera detenido. Para lograrlo, el estado de la CPU (el "contexto") debe ser exactamente el mismo antes y después de la interrupción.
- Los registros de propósito general contienen datos cruciales para la ejecución del programa. La rutina de interrupción necesita usar estos registros para sus propias operaciones. Si no se guardan sus valores, la rutina los sobrescribiría, y el programa original no podría continuar correctamente, lo que llevaría a errores o fallos del sistema.
- El registro `EPC` es el "marcador de posición" que le dice al procesador dónde debe reanudar la ejecución. Si no se guardara la dirección de retorno en este registro, la CPU no tendría forma de saber a dónde volver después de terminar con la interrupción.
- El registro `Status` contiene información crítica sobre el estado del procesador, como el modo de operación y la habilitación de interrupciones. La rutina de interrupción debe operar en un estado controlado (modo *kernel* y, a menudo, con las interrupciones deshabilitadas) para evitar problemas. Guardar y restaurar el `Status` asegura que el procesador vuelva al estado correcto para el programa de usuario.

10 Interrupciones de reloj y control de ejecución

10.1 a) Uso de la interrupción de reloj para el control de ejecución

1. **Para evitar que un programa quede en un bucle infinito:**
   - **Mecanismo:** En un sistema operativo multitarea, el *scheduler* asigna a cada proceso una "porción de tiempo" o *quantum* para que se ejecute. El sistema operativo programa el temporizador de *hardware* para que genere una interrupción cuando este tiempo haya expirado.
   - **Proceso:** Si un programa entra en un bucle infinito, monopolizará el procesador. Sin embargo, la interrupción de reloj ocurrirá al final del *quantum*. Esta interrupción fuerza al procesador a detener el bucle infinito y saltar al manejador de interrupciones del sistema operativo. Una vez allí, el sistema operativo puede realizar un cambio de contexto (salvar el estado del programa en bucle y cargar el estado de otro programa), permitiendo que otros procesos se ejecuten. De esta forma, el programa en bucle no congela todo el sistema, y el sistema operativo mantiene el control.
2. **Para finalizar programas que superan un tiempo máximo de ejecución:**
   - **Mecanismo:** Un sistema operativo puede imponer un límite de tiempo de ejecución a un programa, útil para evitar que programas maliciosos o con fallos consuman recursos indefinidamente. Este tiempo máximo es significativamente mayor que el *quantum* normal.
   - **Proceso:** El sistema operativo establece un temporizador para generar una interrupción cuando el tiempo máximo de ejecución del programa haya transcurrido. Si el programa sigue en ejecución cuando ocurre esta interrupción, el manejador de interrupciones del sistema operativo se da cuenta

de que ha expirado el tiempo límite. En lugar de simplemente realizar un cambio de contexto, el sistema operativo puede tomar medidas para terminar el proceso de forma forzada, liberar sus recursos y notificar al usuario.

10.2 b) ¿Qué debe hacer el software si el programa finaliza antes de que ocurra la interrupción de reloj?

Si un programa finaliza de manera normal antes de que se genere la interrupción de reloj que estaba programada para él, el software del sistema operativo debe realizar una acción crucial para evitar errores futuros.

El programa que finaliza lo hace a través de una llamada al sistema (`syscall`). Esta llamada genera una excepción que transfiere el control al sistema operativo de manera síncrona.

Cuando el sistema operativo tome el control, debe hacer lo siguiente:

1. **Cancelar la interrupción pendiente del temporizador:** El sistema operativo sabe que el programa que iba a ser monitoreado por el temporizador ha terminado. Por lo tanto, debe escribir en el registro de control del temporizador de *hardware* para deshabilitar o *resetear* la interrupción pendiente que se había programado para ese proceso.

2. **Evitar interrupciones falsas:** Si el sistema operativo no cancelara la interrupción, esta se generaría más tarde, cuando el temporizador llegara a cero. En ese momento, otro programa podría estar en ejecución. La interrupción de reloj haría que el sistema operativo saltara al manejador de interrupciones y, al comprobar el estado del sistema, podría intentar procesar una interrupción que ya no es relevante o para un programa que ya no existe, lo que podría causar un fallo.

11 Análisis y Discusión de los Resultados

Los dos algoritmos son ejemplos del funcionamiento del sondeo. Muestran de una manera funcional y simple como se gestiona la comunicación con periféricos. Sin embargo, su verdadera utilidad radica en que, al mismo tiempo, ilustran el costo que conlleva esta simplicidad: una ineficiencia alta del uso de la CPU y la imposibilidad de realizar tareas concurrentes.

En un sistema informático real, donde el tiempo del procesador es el recurso más valioso, el manejo de E/S debe ser asíncrono. Los resultados de los dos programas nos dan una idea clara de por qué los sistemas modernos se basan en interrupciones, que permiten que el procesador dedique su tiempo a tareas productivas y solo atienda a los periféricos cuando estos lo solicitan de forma activa.