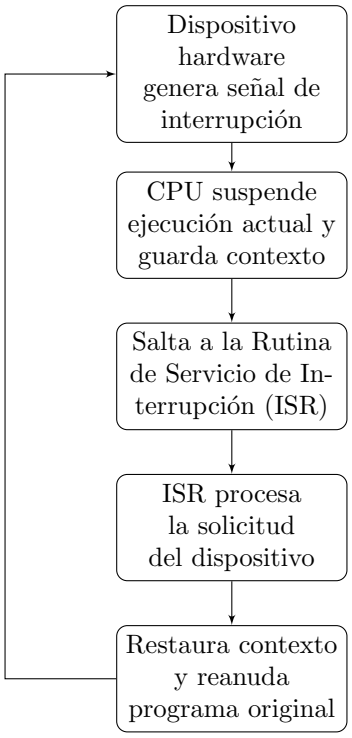
Informe Practica 4

Samuel Primera C.I: 31.129.684, Samuel Reyna CI: 30.210.759

29 de Julio 2025

1. Explique con un diagrama cómo funciona el ciclo de una interrupción de hardware



El ciclo de una interrupción de hardware sigue estos pasos:

1. Un dispositivo hardware genera una señal de interrupción.
2. La CPU suspende la ejecución del programa actual, guarda su estado (contexto), y salta a la rutina de servicio de interrupción (ISR).
3. El ISR procesa la solicitud del dispositivo.

4. Una vez completado el ISR, la CPU restaura el contexto guardado y reanuda la ejecución del programa original.

2. ¿Qué diferencias hay entre gestionar E/S con sondeo y hacerlo con interrupciones?

Aspecto	Gestión con sondeo (Polling)	Gestión con interrupciones
Mecanismo de operación	El procesador verifica repetidamente el estado del dispositivo de E/S.	El dispositivo de E/S notifica al sistema operativo cuando termina una operación o hay un error.
Uso del procesador	Consume ciclos de CPU en espera activa, reduciendo la productividad.	Permite al procesador realizar otras tareas mientras espera, mejorando la eficiencia.
Acceso al dispositivo	Cada verificación requiere un acceso separado al dispositivo de E/S.	Solo se accede al dispositivo cuando este genera una interrupción.
Latencia y eficiencia	Ineficiente para operaciones lentas, ya que el procesador queda ocupado verificando.	Más eficiente, especialmente con E/S lentas, ya que evita esperas innecesarias.
Respuesta a eventos	Depende de la frecuencia de sondeo; puede haber retrasos en detectar cambios.	Respuesta inmediata, ya que el dispositivo avisa activamente.
Complejidad	Más simple de implementar, pero menos escalable.	Más complejo (manejo de interrupciones), pero más escalable y eficiente.
Escenario típico	Útil cuando el tiempo de E/S es predecible y corto.	Ideal para dispositivos lentos o sistemas donde la eficiencia del CPU es crítica.

3. ¿Qué ventajas tiene el uso de interrupciones en términos de uso del procesador?

El uso de interrupciones ofrece varias ventajas significativas en términos de eficiencia y uso del procesador:

- **Mayor eficiencia del procesador:** Las interrupciones permiten al procesador evitar la espera activa (*polling*) por la finalización de operaciones de E/S

lentas. En lugar de que el procesador dedique ciclos valiosos a comprobar repetidamente el estado de los dispositivos de E/S, puede ejecutar otras tareas o programas.

- **Multitarea y concurrencia:** Cuando una operación de E/S solicita datos, el procesador puede cambiar a la ejecución de otra tarea. Una vez que la operación de E/S se completa y genera una interrupción, el procesador es notificado y puede volver a la tarea original o a una tarea relacionada. Esto es crucial para la multitarea y para sistemas donde múltiples programas necesitan compartir recursos.
- **Respuesta a eventos asíncronos:** Las interrupciones son el mecanismo ideal para que el hardware notifique al procesador sobre eventos asíncronos (eventos que no están directamente ligados a la ejecución de una instrucción específica en el programa en curso), como la finalización de una transferencia de datos o un error de hardware.
- **Mejora del tiempo de respuesta (latencia):** Aunque no siempre minimizan la latencia para una única operación de E/S (como podría hacerlo un sondeo muy eficiente en un sistema simple), en un entorno con múltiples tareas o dispositivos, las interrupciones permiten al sistema responder de manera más oportuna a diversos eventos, mejorando la percepción general de la interactividad y eficiencia del sistema.

4. ¿Qué registros especiales se utilizan en MIPS32 para gestionar interrupciones?

En un sistema MIPS32, el coprocesador 0 es el encargado de gestionar las excepciones e interrupciones. Para ello, se utilizan varios registros especiales:

- **Registro Causa (Cause):** Este registro proporciona información sobre la causa específica de la excepción o interrupción. Contiene bits que el hardware modifica para señalar el tipo de evento que la ha provocado.
- **Registro Estado (Status):** Este registro, junto con el registro Causa, proporciona información sobre el estado del sistema de procesamiento de la interrupción. Contiene bits que controlan el estado de las interrupciones, como la habilitación o deshabilitación de las mismas.
- **Registro EPC (Exception Program Counter):** Este registro almacena la dirección de la instrucción que causó la excepción o interrupción. Esto permite que, una vez que la rutina de servicio ha manejado la interrupción, el programa pueda reanudar su ejecución desde el punto correcto.
- **Registro BadVAddr (Bad Virtual Address):** Este registro almacena la dirección virtual involucrada en un fallo de acceso a memoria, lo cual es relevante en el manejo de excepciones relacionadas con la memoria, como los fallos de página.

5. ¿Por qué es necesario guardar el contexto al entrar en una rutina de servicio?

Es necesario guardar el contexto (especialmente los registros del procesador) al entrar en una rutina de servicio por varias razones fundamentales, principalmente para asegurar la correcta reanudación del programa interrumpido y para el funcionamiento adecuado de la rutina de servicio misma:

- **Preservación del estado del programa interrumpido:** Cuando ocurre una interrupción, el procesador deja de ejecutar el programa actual para atender la rutina de servicio. Si la rutina de servicio utiliza los mismos registros que el programa que fue interrumpido, sus valores se sobrescribirían. Al guardar los valores de los registros del programa original en memoria (generalmente en la pila), se asegura que el estado del programa no se pierda.
- **Reanudación correcta de la ejecución:** Una vez que la rutina de servicio completa su tarea, el programa original debe reanudar su ejecución exactamente desde el punto donde fue interrumpido y con su estado intacto. Restaurar los registros previamente guardados desde la memoria permite que el programa continúe como si la interrupción nunca hubiera ocurrido, desde su perspectiva funcional.
- **Independencia de la rutina de servicio:** Al guardar el contexto, la rutina de servicio puede utilizar los registros del procesador sin preocuparse por sobrescribir datos importantes del programa que la invocó. Esto facilita la programación de las rutinas de servicio, ya que no necesitan saber qué registros está utilizando el programa principal.

6. Momentos en que pueden generarse excepciones en un sistema MIPS32

a) Enumera al menos 4 situaciones en las que se pueda generar una excepción (por ejemplo: desbordamiento aritmético, fallo de dirección, etc.).

- **Desbordamiento aritmético (overflow):** Ocurre en operaciones como `add` cuando el resultado excede el rango de 32 bits.
- **Fallo de dirección (address error):** Acceso a memoria con dirección no alineada (ej: `lw` o `sw` en dirección que no es múltiplo de 4).
- **Instrucción no implementada:** El hardware no reconoce el opcode de la instrucción.
- **Interrupción externa:** Señal de un dispositivo de E/S que requiere atención inmediata.

b) Explica qué etapas del pipeline pueden provocar una excepción y por qué.

- **Etapa IF (Fetch):**

- **Causas:**
  - Fallo al leer la instrucción (dirección inválida o no alineada).
  - Acceso a memoria protegida o no disponible.

- **Etapa ID (Decode):**
  - **Causas:**
    - Instrucción con opcode inválido o no soportado.
    - Intento de usar un registro no existente (en algunos diseños).
- **Etapa EX (Execute):**
  - **Causas:**
    - Desbordamiento en operaciones aritméticas (ej: `add`, `sub`).
    - División por cero (depende de la implementación).
- **Etapa MEM (Memory Access):**
  - **Causas:**
    - Dirección de memoria no alineada en load/store (ej: `lw` en 0x1001).
    - Violación de permisos (ej: escritura en memoria de solo lectura).
- **Etapa WB (Writeback):**
  - Normalmente no genera excepciones, ya que solo escribe resultados en registros.

Nota adicional:

- Las excepciones se manejan mediante el **coprocesador 0 (CP0)** en MIPS32.
- El flujo del pipeline se interrumpe y el PC salta al **vector de excepciones** (ej: `0x80000180`).

7. Estrategias de tratamiento de excepciones e interrupciones

a) Diferencias entre interrupciones y excepciones (¿son síncronas o asíncronas?)

- **Excepciones:**
  - Son eventos **síncronos**. Esto significa que están directamente relacionadas con la ejecución de una instrucción específica y ocurren de manera predecible en el mismo punto del programa si las condiciones que las causan se repiten.
  - Son generadas por el propio procesador al detectar un problema o una condición especial durante la ejecución de una instrucción.

- **Interrupciones:**
  - Son eventos **asíncronos**. Esto significa que no están directamente relacionadas con una instrucción específica que se está ejecutando en el momento en que ocurren. Pueden ocurrir en cualquier momento.
  - Son generadas por eventos externos al procesador, como la finalización de una operación de E/S o un temporizador.

b) Estrategias para tratar excepciones en un sistema MIPS32

1. Manejo general de excepciones (no vectorizado)

- Para la mayoría de las excepciones, MIPS utiliza un único punto de entrada para el manejador de excepciones. El procesador salta a una dirección predefinida, que es `0x80000180` (en hexadecimal) en el espacio de direcciones del núcleo.
- Una vez allí, el sistema operativo descodifica el registro `Cause` para determinar la razón específica de la excepción. Este registro contiene un campo que indica la causa de la excepción.

2. Manejo de fallos de TLB (con punto de entrada especializado)

- Aunque no es una interrupción vectorizada completa en el sentido tradicional, MIPS utiliza una dirección específica para los fallos de la TLB (*Translation Lookaside Buffer*) para mejorar el rendimiento.
- Cuando ocurre un fallo de TLB, el control se transfiere a la dirección `0x80000000` (en hexadecimal). Esto permite un procesamiento más rápido del caso frecuente de fallo de TLB, con una penalización menor para el caso infrecuente de un fallo de página verdadero.

Función del registro EPC (*Exception Program Counter*)

- El registro EPC (*Exception Program Counter*) es un registro de 32 bits que almacena la dirección de la instrucción que causó la excepción.
- Su función principal es permitir que el programa continúe su ejecución después de que el manejador de excepciones haya realizado las acciones necesarias.
- Normalmente, se guarda la dirección de la instrucción + 4, por lo que la rutina de gestión de la excepción debe restar 4 para obtener la dirección correcta de la instrucción infractora.
- El manejador de excepciones utiliza la instrucción `eret` para retornar el flujo de control del programa a la dirección apuntada por el registro EPC, reanudando así la ejecución.

- En el caso de interrupciones externas, la instrucción no habrá comenzado su ejecución cuando se guarda su dirección en el EPC.

8. Habilitación de interrupciones en dispositivos y procesador

a) Cómo se habilitan las interrupciones

En el teclado:

- El teclado (un dispositivo receptor) tiene un **Registro de Control del Receptor** en la dirección $\texttt{ffff0000}_{\texttt{hex}}$.
- Para habilitar interrupciones del teclado, el **bit 1** de este registro (conocido como *"permitir interrupción"* o *"interrupt enable"*) debe ser activado a `1` por un programa.
- Cuando se escribe un carácter en el teclado y este bit está activado, el terminal solicita una interrupción de hardware de **nivel 1**.

En la pantalla:

- La pantalla, como dispositivo de salida, se controla a través de un **Registro de Control del Transmisor** en la dirección $\texttt{ffff0008}_{\texttt{hex}}$.
- Similar al teclado, el **bit 1** de este registro (*"permitir interrupción"*) debe ser activado a `1` por un programa.
- Cuando el transmisor está listo para aceptar un nuevo carácter de salida (es decir, su bit *"preparado"* se pone a `1`), el terminal solicitará una interrupción de hardware de **nivel 0**.

En el procesador:

- El procesador MIPS utiliza el registro **Status** para controlar la habilitación de interrupciones. Este registro contiene varios campos clave:
  - **Bit IE** (*Interrupt Enable - Habilitación de Interrupción*): Este es un bit global. Si está en `0`, ninguna interrupción puede afectar al procesador. Para permitir interrupciones, debe estar en `1`.
  - **Campo "Máscara de interrupción"** (*Interrupt Mask*): Este campo contiene bits individuales (uno para cada nivel de interrupción de hardware y software). Para que una interrupción específica sea atendida, el bit correspondiente en la máscara debe estar en `1`. Esto permite un control más refinado sobre qué interrupciones se procesan.

- **Bit EL** (*Exception Level - Nivel de Excepción*): Normalmente está en `0`. Cuando ocurre una excepción, se activa a `1`. Si está en `1`, deshabilita nuevas interrupciones y evita que el registro `EPC` se actualice, protegiendo el manejador de excepciones de ser perturbado. Debe ser restablecido por el manejador.

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

- Si las interrupciones están habilitadas en los dispositivos (es decir, el dispositivo está configurado para generar una señal de interrupción), pero no en el procesador, el procesador **no responderá** a esas interrupciones.
- El dispositivo podría activar su bit de interrupción pendiente en el registro `Cause` del coprocesador 0 del MIPS.
- Sin embargo, si el bit de *"permiso de interrupción"* en el registro `Status` del procesador está en `0`, o si el bit correspondiente en la máscara de interrupción para ese nivel de prioridad está en `0`, el procesador simplemente **ignorará** la señal de interrupción y continuará su ejecución normal.
- El procesador solo se interrumpirá si, posteriormente, el bit de máscara o el bit de habilitación global de interrupciones se activan.

9. Procesamiento de interrupciones

a) Paso a paso cuando se produce una interrupción de reloj

1. Ocurrencia de la Interrupción

- El timer del sistema genera una señal de interrupción al alcanzar un valor predefinido (por ejemplo, cuando el contador llega a cero).

2. Acciones Inmediatas del Hardware

- **Verificación de interrupciones:**
  - El CPU comprueba si las interrupciones están habilitadas (bit IE en el registro `Status`).
  - Si IE = 1 y no hay otra excepción en curso (EXL = 0), el hardware procede.
- **Procesamiento inicial:**
  - Guarda el registro `EPC` (*Exception Program Counter*): Almacena la dirección de la siguiente instrucción a ejecutar.
  - Actualiza el registro `Status`:

- Establece EXL = 1 (modo kernel, bloqueando nuevas excepciones).
- Desactiva interrupciones (IE = 0).

- Configura el registro `Cause`:
  - Asigna el código de excepción (6 para interrupción de reloj).
- Salto al manejador de interrupciones:
  - Carga la dirección del vector de excepciones (`0x80000180` en MIPS32) en el `PC`.

3. Ejecución de la Rutina de Servicio (Software)

El sistema operativo o manejador de interrupciones realiza las siguientes acciones:

a) Almacenamiento del Contexto

- Guarda los registros en la pila del kernel (el hardware solo guarda `EPC` y modifica `Status/Cause`):
  - Registros de propósito general (`$at, $v0, $a0, $t0, ..., $ra`, etc.).
  - Registros `Status` y `Cause` (para su posterior restauración).
  - Cualquier otro registro susceptible de modificación.

b) Procesamiento de la Interrupción

- **Reinicio del timer:**
  - Establece un nuevo valor en el registro del timer para el siguiente ciclo.
- **Gestión del planificador** (en sistemas multitarea):
  - Reduce el quantum del proceso actual.
  - Decide si es necesario un cambio de contexto (si el quantum se ha agotado).
    - En caso afirmativo, guarda el estado del proceso actual (en su PCB) y carga el siguiente.

c) Restauración del Contexto

- Recupera los registros guardados desde la pila del kernel.
- **Preparación para el retorno:**
  - Asegura que el registro `Status` tenga los valores correctos (EXL = 0, IE restaurado).

4. Retorno de la Interrupción (Instrucción ERET)

- El software ejecuta `ERET` (*Exception Return*):
  - Restaura el `PC` desde `EPC`: Reanuda la ejecución del programa interrumpido.
  - Retorno al modo usuario: Establece EXL = 0.
  - Reactivación de interrupciones: Si estaban habilitadas previamente (IE = 1).

b) ¿Por qué es importante guardar el contexto (registros generales, EPC, Status) al entrar en la rutina?

Es crítico guardar el contexto (registros generales, `EPC`, `Status`) al entrar en la rutina de manejo de interrupciones por varias razones fundamentales:

- **Reanudación Correcta del Programa Interrumpido:** La interrupción es un evento inesperado. Para que el programa interrumpido pueda reanudarse de manera transparente y correcta desde el punto donde fue interrumpido, necesita que su estado original (valores de todos los registros) sea restaurado con exactitud. El `EPC` es fundamental para saber la dirección de la instrucción a retomar, y los registros de propósito general contienen los datos y direcciones con los que el programa estaba operando.
- **Aislamiento y Protección de Procesos:** En un sistema con múltiples procesos, la interrupción de reloj es comúnmente utilizada para la gestión de tiempo compartido y la planificación de procesos. Guardar el contexto asegura que el sistema operativo puede pausar un programa de usuario, ejecutar su propia lógica (del sistema operativo), y luego cambiar a otro programa o volver al mismo, sin que los programas interfieran entre sí ni que la información del proceso interrumpido se corrompa.
- **Manejo Robusto de Interrupciones:** Algunas interrupciones pueden ocurrir mientras el sistema operativo ya está manejando otra interrupción (anidamiento de interrupciones). Si el contexto no se guarda, una segunda interrupción podría sobrescribir el `EPC` o el registro de causa de la primera, haciendo imposible retornar a la instrucción original y resultando en la pérdida de información de estado crítico. Guardar el contexto y controlar la habilitación/deshabilitación de interrupciones previene estos desastres.
- **Diagnóstico de la Interrupción:** El registro `Cause` es esencial para que el software del sistema operativo pueda determinar la razón de la interrupción. El registro `Status` es clave para gestionar la prioridad y el enmascaramiento de las interrupciones.
- En resumen, guardar el contexto permite que el sistema operativo gestione las interrupciones de forma segura y eficiente, garantizando la integridad de los programas y la estabilidad general del sistema.

10. Interrupciones de reloj y control de ejecución

a) Uso de una interrupción de reloj para controlar la ejecución de programas

- Para evitar que un programa quede en un bucle infinito y finalizar programas que superan un tiempo máximo de ejecución, se puede utilizar una interrupción de reloj.
- Un registro de contador (`Count`) en el procesador (como el del coprocesador 0 en MIPS) se incrementa a una frecuencia fija.
- Cuando el valor de este contador iguala un valor predefinido en un registro de comparación (`Compare`), se produce una interrupción de hardware.
- Esta interrupción es una excepción, un evento no previsto que interrumpe la ejecución normal del programa y transfiere el control al sistema operativo (SO) en una dirección predefinida (por ejemplo, $\texttt{0x80000180}_{\texttt{hex}}$ en MIPS para el manejador de excepciones).
- El SO utiliza el registro de `Causa` para determinar el motivo de la excepción. Si la causa es una interrupción de temporizador, el SO puede entonces tomar la acción apropiada, que incluye detener la ejecución del programa que ha excedido su tiempo asignado. Esto asegura que un programa no se ejecute indefinidamente ni monopolice el procesador.

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

Si un programa finaliza su ejecución de forma voluntaria (por ejemplo, mediante una llamada al sistema como `exit` o alcanzando su fin) antes de que se produzca la interrupción de reloj programada para su rodaja de tiempo", el software (sistema operativo) debe:

- **Tomar el control inmediatamente:** Una llamada al sistema (`syscall`) para terminar el programa transferirá el control al sistema operativo.
- **Liberar recursos y limpiar el proceso:** El sistema operativo procederá a limpiar todos los recursos asignados a ese programa (memoria, archivos abiertos, etc.).
- **Ignorar o reconfigurar la interrupción pendiente:** La interrupción de reloj ya programada para ese "slice"de tiempo se vuelve irrelevante para ese proceso. El sistema operativo, al haber tomado el control, no permitirá que el temporizador (`Count/Compare`) desencadene incorrectamente un cambio de contexto a un proceso ya terminado. Al terminar el proceso y programar el siguiente, el sistema operativo gestiona el temporizador

para el nuevo contexto. Los manejadores de interrupción están diseñados para verificar el contexto actual y la causa de la interrupción antes de tomar acciones, de modo que una interrupción de reloj para un proceso ya finalizado sería manejada sin intentar reanudar un programa que no existe.

11. Análisis y Discusión de los Resultados

Esta práctica se caracterizó por el uso de interrupciones por hardware para la gestión de entradas/salidas y la implementación de rutinas de servicio. El desarrollo incluyó dos algoritmos principales con enfoques distintos pero complementarios.

En el primer algoritmo, se comenzó con la declaración de variables, incluyendo un intervalo temporal de 20 segundos y la dirección de memoria asignada al teclado. En la función principal se inicializaron los registros necesarios. Luego, en el bucle principal se implementa una espera activa de `50 ms` para optimizar el uso de la CPU. Este bucle verifica las entradas por teclado mediante la lectura del bit de *ready* en el registro de control. Cuando se detectaba una entrada válida, el sistema procede a leer y almacenar el carácter en un *buffer* circular, actualizando adecuadamente los punteros `head` y `tail` con su correspondiente manejo de *wrap-around* para garantizar el funcionamiento correcto del *buffer*.

La gestión del tiempo se realiza mediante la rutina `check_time`, que acumula los intervalos de `50 ms`. Cada 20 segundos, el sistema muestra el contenido completo del *buffer* a través de la función `print_buffer`. Esta última función se encarga de imprimir un encabezado y luego recorre el *buffer* carácter por carácter, actualizando adecuadamente el puntero `tail` y manejando los casos especiales de *buffer* vacío o lleno. Todo este proceso demuestra la importancia de las interrupciones de teclado para el correcto funcionamiento del sistema de almacenamiento y visualización de caracteres.

El segundo algoritmo se centra en la visualización gráfica mediante un *Bitmap Display*. La inicialización del sistema incluye la configuración de los registros necesarios y la preparación de los valores de color en formato hexadecimal. El ejercicio consiste en ciclos de visualización controlados por un interruptor simulado mediante teclado. Cada ciclo lee el color de los píxeles, verifica el estado de completado, y gestiona tiempos de espera específicos para cada transición de color. El sistema mantiene un bucle principal que garantiza la secuencia correcta de colores y tiempos, volviendo al estado inicial al completar el ciclo completo.

La implementación de estos algoritmos permitió observar en la práctica varios conceptos fundamentales. En el primer caso, se evidenció la importancia de las interrupciones de hardware para la gestión eficiente de dispositivos de entrada como el teclado. El segundo ejercicio demostró el uso de visualización gráfica y temporizaciones precisas, mostrando cómo interactúan diferentes componentes del sistema. Ambos ejemplos sirvieron para comprender mejor el manejo de *buffers*, la gestión de tiempos y la importancia de las rutinas de servicio de interrupciones.