Informe de LAB 4

Michelle Sayas y Brandon Zanotti

28 de julio de 2025

Resumen

1. Explique con un diagrama cómo funciona el ciclo de una interrupción de hardware (desde que ocurre el evento hasta que se atiende).

1. Ocurre el evento de interrupción

Un dispositivo externo (teclado, temporizador, etc.) activa una señal de interrupción. Esta señal se recibe en la unidad de control (no visible directamente en el diagrama), que monitorea las líneas de interrupción.

2. Verificación de habilitación

La unidad de control verifica si las interrupciones están habilitadas (bit IE del registro Status en el CP0). También revisa las máscaras de interrupción (IM) y el registro Cause para identificar la fuente.

3. Guardado del estado

Se guarda el valor del PC (program counter) en el registro 'EPC' (Exception Program Counter) del CP0. Esto permite volver al punto donde fue interrumpido el programa. También se actualiza el registro 'Cause' con el tipo de excepción/interrupción.

4. Cambio del flujo del programa

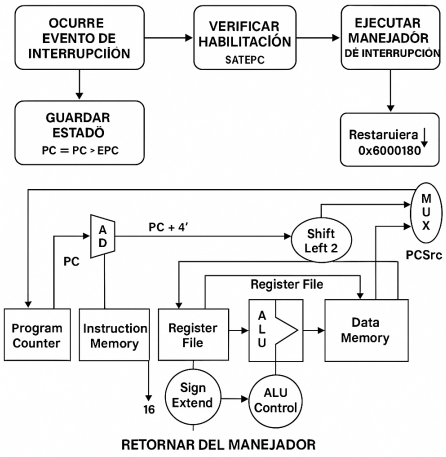
El PC se redirige a una dirección fija de excepción/interrupción, normalmente: 0x80000180 (modo kernel, interrupción general en MIPS). Esto modifica el flujo normal del diagrama que compartiste, desviando el valor del PC que normalmente saldría de la suma PC + 4 y enviándolo al handler de interrupción.

5. Ejecución del manejador de interrupción (ISR)

Se ejecuta el código en esa dirección, que generalmente guarda registros necesarios, atiende el evento (leer buffer, resetear hardware, etc.). Este código accede a memoria e instrucciones igual que en tu diagrama, solo que el flujo de instrucciones proviene del handler.

6. Retorno del manejador

Tras manejar la interrupción, se usa la instrucción ERET (Exception Return). Esto hace que el PC recupere el valor guardado en EPC, volviendo al flujo normal del programa justo donde fue interrumpido.



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

Caracteristicas	Sondeo(Polling)	Interrupciones
Funcionamiento	El procesador revisa periódicamente el estado del dispositivo de E/S.	El dispositivo de E/S interrumpe al procesador cuando necesita atención.
Uso del PC	Desperdicia ciclos revisando constantemente	Libera al CPU hasta que se requiere su atención.
Eficiencia	Baja eficiencia, especialmente con dispositivos lentos	Alta eficiencia, permite multitarea
Implementación	Más sencilla a nivel de hardware/software.	Requiere lógica más compleja (vector de interrupciones, handler, etc.)
Tiempo de respuesta	Puede haber retardo si el sondeo es infrecuente	Generalmente rápido, responde al momento del evento.

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

Mayor eficiencia del procesador: El CPU no pierde tiempo revisando dispositivos constantemente; ejecuta otras tareas mientras espera.

Multitarea: Permite al sistema atender múltiples procesos o eventos de E/S sin bloquear la ejecución del programa principal.

Menor latencia en respuesta: Las interrupciones permiten una reacción inmediata ante eventos externos críticos (como una señal de emergencia).

Mejor aprovechamiento de recursos: El CPU puede entrar en modo bajo consumo o ejecutar código útil en lugar de realizar bucles de espera.

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

En MIPS32, las interrupciones se gestionan a través del Coprocesador 0 (CP0), el cual incluye registros especializados para controlar, identificar y atender interrupciones.

Regsitro CP0	Nombre	Función
12	Status	Controla si las interrupciones están habilitadas (bit IE), y qué tipos están permitidas (IM)
13	Cause	Indica la causa de la excepción/interrupción (campo IP, ExcCode)
14	EPC (Exception Program Counter)	Guarda la dirección del PC donde ocurrió la interrupción.
8	BadVAddr	Guarda la dirección inválida en caso de fallos de memoria (no siempre usado para interrupciones).

El ERET (Exception Return) es la instrucción que restaura el PC desde el registro EPC, devolviendo el control al programa principal después de atender la interrupción. Se debe usar al final del manejador de interrupción.

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

Es necesario guardar el contexto (registros) al entrar en una rutina de servicio de interrupción (ISR, Interrupt Service Routine) para preservar el estado del programa interrumpido. El contexto se refiere a los valores actuales de los registros del procesador, incluyendo:

El PC (Program Counter), los registros generales ($t0$,s1, etc.), el registro de estado (Status en MIPS), el registro HI/LO (si se usan operaciones de multiplicación/división) y otros registros especiales como EPC y Cause en MIPS.

Por qué es necesario guardar el contexto?

1. Preservar la ejecución del programa interrumpido.

Cuando ocurre una interrupción, el procesador suspende temporalmente el programa actual para ejecutar el código del manejador. Si no se guardan los registros:

Se sobrescriben los datos actuales del programa principal.

Al retornar del ISR, el programa puede comportarse de forma incorrecta o impredecible.

2. Separación entre ISR y programa principal.

El handler de interrupción puede usar los mismos registros que el programa principal.

Guardar y restaurar los registros garantiza que el programa principal no se vea afectado por las operaciones del ISR.

3. Mantenimiento del flujo de control.

El valor del PC debe guardarse en EPC (en MIPS), para que el retorno con ERET pueda reanudar la ejecución exactamente donde fue interrumpido.

En resumen, guardar el contexto permite al procesador "pausarüna tarea, atender otra urgente, y luego continuar como si nada hubiera pasado. Esto es clave para sistemas seguros, estables y deterministas, especialmente en entornos de tiempo real o multitarea.

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

a) Situaciones en las que se puede generar una excepción en MIPS32

Una excepción en MIPS32 es un evento que interrumpe el flujo normal del programa debido a una condición inesperada, ya sea por error o por necesidad del sistema (como una syscall).

Ejemplos de situaciones que generan excepciones:

Desbordamiento aritmético (Overflow)

Ocurre cuando el resultado de una operación (como add) excede el rango representable por los registros.

Solo se genera en instrucciones aritméticas con verificación de desbordamiento (add, no addu).

Acceso inválido a memoria

Por ejemplo, acceso a una dirección no alineada en instrucciones lw, sw, lh, etc.

También ocurre si se accede a direcciones no mapeadas o protegidas.

Instrucción no válida (Reserved Instruction)
Se intenta ejecutar una instrucción que no está definida o habilitada para el modo actual (usuario/kernel).
Llamada al sistema (Syscall)
Se ejecuta la instrucción syscall para solicitar servicios al sistema operativo.
No es un error, pero requiere atención especial (manejador).

División por cero (en algunos sistemas con extensiones)
Si se divide entre cero usando operaciones que lo permiten (div), puede generar excepción.

Breakpoint
La instrucción break detiene la ejecución para depuración (debug).

b) Etapas del pipeline que pueden provocar una excepción y por qué
MIPS32 tiene un pipeline clásico de 5 etapas:

Etapa	Excepciones posibes
IF (Instruction Fetch)	Fallo al acceder a una dirección inválida de instrucción.
ID (Instruction Decode)	Instrucción no válida (no reconocida), syscall o break.
EX (Execute)	Desbordamiento aritmético (add), División por cero (si aplica).
MEM (Memory Access)	Acceso a dirección no válida o no alineada, Fallo de segmentación o página.
WB (Write Back)	Generalmente no produce excepciones en esta etapa.

Explicación:

a) IF (Instruction Fetch):
Si el PC contiene una dirección inválida o no alineada, se genera una excepción al intentar buscar la instrucción.
b) ID (Decode):
Se detectan instrucciones inválidas (no reconocidas). Se reconocen instrucciones especiales como syscall y break.
c) EX (Execute):
Se realizan operaciones aritméticas que pueden causar overflow. Se detectan divisiones por cero si se implementan.
d) MEM (Memory):
Aquí se generan errores por direcciones de memoria ilegales o accesos no alineados, tanto en lectura como en escritura.

En resumen, las excepciones permiten que el sistema detecte y maneje errores o eventos especiales. Entender dónde y por qué pueden generarse es esencial para diseñar software robusto y predecible en MIPS32.

7. Estrategias de tratamiento de excepciones e interrupciones

a) Diferencias entre interrupciones y excepciones

Caracteristicas	Interrupciones	Excepciones
Origen	Externas al procesador (dispositivos de E/S, temporizador).	Internas al procesador (errores o eventos del programa).
Sincronia	Asíncronas: pueden ocurrir en cualquier momento.	Síncronas: ocurren en un punto predecible, durante la ejecución de una instrucción.
Ejemlos	Teclado, reloj, llegada de datos.	Desbordamiento aritmético, syscall, acceso inválido a memoria.
Control de Sitemas	El sistema debe estar preparado para eventos impredecibles.	Se relacionan con el flujo actual del programa.
Dependencia de programa	Independientes del programa que se ejecuta	Generadas directamente por el código en ejecución.

b) Estrategias para tratar excepciones en MIPS32

En MIPS32, las excepciones se gestionan mediante el Coprocesador 0 (CP0). Aquí van dos estrategias básicas de tratamiento:

b1. Redirección fija a una dirección base
Cuando ocurre una excepción, el hardware automáticamente:
Guarda la dirección del PC en el registro EPC.
Guarda la causa en el registro Cause.
Redirige la ejecución a una dirección fija: 0x80000180 (modo kernel, excepción general).

Ventajas: Simple de implementar.
Desventajas: Requiere lógica adicional para determinar el tipo de excepción manualmente.

b2. Vectores de excepción personalizados (opcional con configuración EBase)

En algunas versiones de MIPS se permite configurar una base distinta (registro EBase) y usar vectores diferentes según el tipo de excepción.

Permite tener múltiples rutinas especializadas (mejor desempeño y mantenimiento).

Ventaja: mejora la eficiencia y organización.

Desventaja: requiere soporte adicional y configuración más compleja.

¿Cómo se redirige la ejecución hacia la rutina de servicio?

+ Se detecta la excepción/interrupción.
+ El hardware:
- Guarda el PC actual en el registro EPC.
- Coloca un código de causa en el registro Cause.
- Redirige la ejecución al manejador de excepciones (por defecto, 0x80000180).
+ Se ejecuta el código de atención (rutina de servicio).
+ Al finalizar, se ejecuta la instrucción ERET (Exception Return), que:
-Recupera el PC desde EPC.
-Restaura el flujo normal del programa.

Función del registro EPC (Exception Program Counter)

EPC guarda la dirección de la instrucción que causó la excepción. Es fundamental para que, tras atender la excepción o interrupción, el procesador pueda retomar la ejecución correctamente. Al usar ERET, el procesador salta nuevamente a la dirección almacenada en EPC, reanudando el programa como si nada hubiera pasado.

8. Habilitación de interrupciones en dispositivos y procesador

a) ¿Cómo se habilitan las interrupciones en:

a1. El teclado (dispositivo de entrada): El controlador del teclado (por ejemplo, un periférico conectado vía memoria mapeada) tiene un registro de control.

Para habilitar interrupciones:

Se debe escribir un valor específico (por ejemplo, 1 en un bit INT-EN) en el registro de control del teclado.

Esto indica que el teclado debe enviar una interrupción al procesador cuando una tecla es presionada.

a2. La pantalla (por ejemplo, controlador VGA o framebuffer): La pantalla generalmente no genera interrupciones frecuentemente, salvo en situaciones como:

Fin de una operación DMA (transferencia directa a memoria).
Vertical blanking interval (para sincronizar).
Para habilitarlas:
Se activa el bit de interrupción correspondiente en el registro de control del dispositivo de vídeo.

a3. El procesador (MIPS32): Para que el procesador responda a interrupciones, se deben configurar correctamente dos niveles de habilitación en el registro Status (registro 12 de CP0):

Bit	Nombre	Función
IE	Interrupt Enable (bit 0)	Habilita globalmente las interrupciones (1 = activadas).
EXL	Exception Level (bit 1)	Si está en 1, el procesador ignora interrupciones (modo excepción).
IM	Interrupt Mask (bits 8-15	Enmascaran o permiten interrupciones de diferentes niveles o fuentes

Ejemplo básico para habilitar interrupciones:
Status.IE = 1 → activar interrupciones globales.
Status.EXL = 0 → permitir atender interrupciones.
Status.IMx = 1 → habilitar la línea específica (por ejemplo, IM7 para el temporizador).

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

Si los dispositivos tienen las interrupciones habilitadas, pero el procesador no, entonces:
El hardware del dispositivo generará señales de interrupción correctamente.
Sin embargo, el procesador las ignorará, porque:
El bit IE = 0 (interrupciones deshabilitadas globalmente), o
El bit EXL = 1 (modo excepción), o
El bit correspondiente en IMx = 0 (máscara activa para esa fuente).
Como resultado, las rutinas de servicio de interrupción (ISR) nunca se ejecutarán, y el sistema no responderá a los eventos externos, aunque hayan ocurrido.

Esto puede llevar a:
Bloqueos si se espera una señal externa (como una tecla).
Fallos en tareas en tiempo real o comunicación.

9. Procesamiento de interrupciones

a) ¿Qué ocurre paso a paso cuando se produce una interrupción de reloj en MIPS32?

a1. Ocurre el evento de interrupción El temporizador (timer) llega a su cuenta límite y genera una señal de interrupción.

Esta señal se envía por una de las líneas de interrupción (por ejemplo, IRQ 7 en Cause.IP7).

a2. El hardware del procesador detecta la interrupción Antes de atender la interrupción, el hardware verifica que:

El bit Status.IE = 1 (interrupciones habilitadas).

El bit correspondiente en Status.IM esté habilitado (por ejemplo IM7).

El bit Status.EXL = 0 (no se está atendiendo ya una excepción).

Si todo está permitido, el procesador interrumpe el flujo actual.

a3. El hardware guarda el contexto mínimo Guarda en CP0.EPC la dirección de la instrucción interrumpida (para poder volver).

Establece el bit Status.EXL = 1 para indicar que está en modo excepción.

Guarda la causa en CP0.Cause, incluyendo qué línea de interrupción fue activada.

a4. El PC se redirige a la rutina de servicio El procesador cambia la dirección de ejecución a 0x80000180, donde se encuentra (o apunta) la rutina de servicio.

El flujo de instrucciones ahora pertenece al código del sistema operativo o manejador de interrupción.

a5. El software (handler) guarda el contexto adicional La rutina de interrupción (escrita en software):

Guarda los registros generales que podría modificar ($t0$–t9, $s0$–s7).

También puede guardar HI/LO, si se van a usar.

Es posible que se guarde el contenido completo del Status, por si se modifica.

a6. El handler atiende el evento Verifica la fuente de interrupción (leyendo Cause).

Ejecuta el código necesario para manejar la interrupción:

Actualiza contadores de reloj.

Despierta procesos bloqueados.

Restaura o planifica tareas (si hay cambio de contexto).

a7. Se restauran los registros guardados El manejador restaura todos los registros guardados al inicio.

Se limpia o reinicia el bit de interrupción en el dispositivo (por ejemplo, reiniciar el timer).

a8. El sistema ejecuta ERET La instrucción ERET (Exception Return) hace que:

El procesador restaure el PC desde EPC.

Se borre el bit EXL para salir del modo excepción.

El flujo normal del programa se reanuda exactamente donde fue interrumpido.

¿Qué registros se guardan y restauran?

Registro	¿Quien lo guarda?	Porque se guarda?
EPC (CP0.14)	hardware	Dirección donde ocurrió la interrupción.
Cause (CP0.13)	hardware	Identifica la causa del evento.
Status (CP0.12)	sotfware (a veces)	Por si se modifica dentro del handler.
Registros $t0–$t9	sotfware	Son temporales y usados por el handler.
Registros $s0–$s7	sotware	Si el handler los modifica, deben restaurarse.
HI/LO	sotfware	Si se usan operaciones aritméticas especiales

b) ¿Por qué es importante guardar el contexto (registros, EPC, Status)?
Porque sin guardar el contexto:

Se perdería la información de la ejecución interrumpida, haciendo imposible reanudarla correctamente.

El handler podría sobrescribir registros que el programa principal necesita, provocando errores lógicos o corrupción de datos.

EPC es necesario para saber dónde reanudar la ejecución.

Status garantiza que se pueda volver al estado exacto anterior (modo usuario/kernel, habilitación de interrupciones, etc.).

Guardar y restaurar el contexto permite que el sistema responda a eventos sin alterar el programa principal.

10. Interrupciones de reloj y control de ejecución

a) ¿Cómo puede usarse una interrupción de reloj para...?

a1. Evitar que un programa quede en un bucle infinito Se puede configurar un temporizador que genere una interrupción periódica (por ejemplo, cada 10 ms). Cada vez que ocurre la interrupción, el sistema operativo puede:

+ Revisar el estado del programa.

+ Llevar una cuenta del tiempo o número de ciclos que ha consumido.

Si un proceso ha estado ejecutándose demasiado tiempo sin pausa, el sistema puede:

+ Forzarlo a ceder el control (cambio de contexto).

+ Detectar que está en un bucle infinito y tomar medidas (detenerlo o reiniciarlo).

Esto evita que un proceso mal escrito bloquee todo el sistema.

a2. Finalizar programas que superan un tiempo máximo de ejecución El sistema operativo puede asignar una cuota de tiempo a cada programa (por ejemplo, 50 ms).Las interrupciones de reloj sirven como un reloj de referencia:

+ Cada interrupción incrementa un contador asociado al proceso.

+ Si el contador alcanza el límite asignado, el sistema puede:

- Terminar el programa automáticamente.

- Registrar un error de "timeout.º matar el proceso.

Esto es esencial en sistemas embebidos, tiempo real o compartición de CPU.

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

El sistema operativo debe cancelar o ignorar la interrupción pendiente, si ya no es necesaria.

Posibles acciones del software:

b1. Liberar o deshabilitar el temporizador:

+ Si el temporizador fue programado solo para controlar ese programa, el sistema puede: Desactivarlo, Reutilizarlo para otro proceso.

b2. Actualizar el planificador:

+ Marcar el proceso como finalizado en la tabla de procesos.

+ Asegurarse de que, cuando ocurra la próxima interrupción del reloj, no se intente acceder al contexto de un programa que ya terminó.

b3. Limpiar recursos:

+ Libera memoria, registros, y estructuras asociadas.

+ Evita fugas de recursos o referencias inválidas.

Esto evita errores como acceder a procesos eliminados o causar fallos por interrupciones mal gestionadas.

En resumen

Situación	Pol del reloj/interrupción
Bucle infinito	Interrupción fuerza revisión periódica del sistema operativo
Tiempo maximo superado	Permite terminar procesos que se exceden
programa termina antes	El sistema debe cancelar o ignorar el temporizador restante

11. Análisis y Discusión de los Resultados

En conclusion, El uso de interrupciones y excepciones en sistemas como MIPS32 es crucial para gestionar eventos que requieren atención inmediata, como interrupciones de dispositivos o errores de ejecución. Las interrupciones son asíncronas y externas, generadas por eventos como un temporizador o un teclado, mientras que las excepciones son síncronas, causadas por errores dentro del flujo del programa, como un desbordamiento aritmético. La correcta habilitación y manejo de interrupciones asegura que los dispositivos puedan comunicar eventos al procesador sin interrumpir su flujo normal de ejecución, mientras que el procesador debe guardar su contexto (registros) al entrar en una rutina de servicio para preservar el estado del programa interrumpido.

En cuanto al control de ejecución y el manejo de interrupciones de reloj, estas permiten al sistema operativo evitar que un programa quede atrapado en un bucle infinito o termine procesos que exceden su tiempo máximo de ejecución, lo cual es fundamental para sistemas multitarea y de tiempo real. El uso adecuado de temporizadores e interrupciones periódicas permite a los sistemas tomar decisiones rápidas sin sobrecargar el procesador. Si un programa finaliza antes de la interrupción, el software debe liberar los recursos asociados y cancelar las interrupciones pendientes, garantizando así que los recursos se gestionen de manera eficiente y sin generar conflictos.