PRÁCTICA DE LABORATORIO 4: INFORME

Jose Contin V29.947.026
Luis Romero V26.729.561

Julio 2025

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

Una interrupción de hardware es una señal que un dispositivo externo (como teclado, mouse, disco, etc.) envía al procesador para indicarle que necesita ser atendido. El procesador detiene temporalmente lo que está haciendo para responder al evento.

Ciclo de una Interrupción de Hardware (paso a paso)

1. Evento de hardware ocurre: Un dispositivo de hardware (ej: teclado, temporizador) genera una señal de interrupción debido a un evento (ej: tecla presionada, timer expirado).
2. El dispositivo genera una señal de interrupción: La señal viaja por las líneas del bus al controlador de interrupciones (PIC o APIC).
3. El controlador de interrupciones evalúa la prioridad: Si la interrupción es válida y tiene mayor prioridad que el proceso actual, la envía al procesador.
4. El procesador interrumpe la ejecución actual: Guarda el estado actual (registros, contador de programa, etc.) para poder retomar luego.
5. El procesador consulta la tabla de vectores de interrupciones: Identifica qué rutina de servicio (ISR) debe ejecutar.
6. Se ejecuta la rutina de servicio de interrupción (ISR): Esta es una función que maneja la causa del evento (leer datos, enviar respuesta, etc.).
7. Finaliza la rutina y se restablece el estado previo: Se restauran los registros y el procesador continúa donde lo dejó.

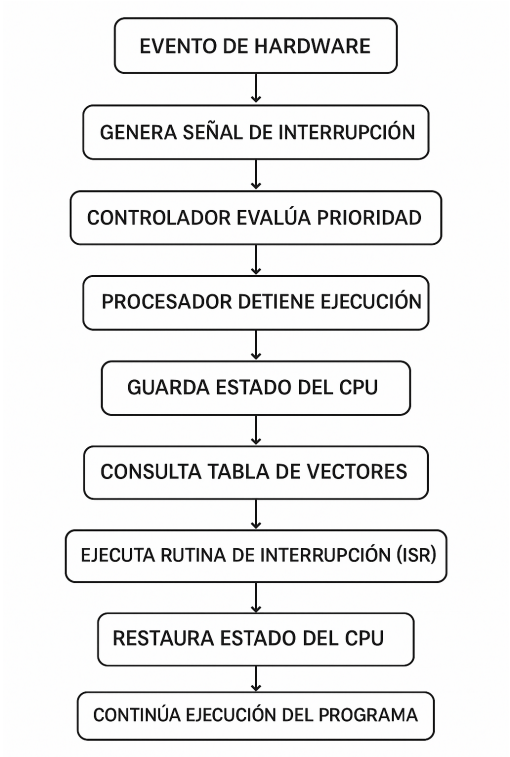

Figure 1: Diagrama de ciclo de una interrupción de hardware.

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

1. Sondeo (Polling)
   - Mecanismo: El procesador revisa constantemente (en un bucle) si un dispositivo necesita atención.
   - Ventajas: Es simple de implementar. No requiere hardware adicional como controladores de interrupciones. Predecible (útil en sistemas de tiempo real crítico).
   - Desventajas: Desperdicia recursos del procesador (ciclo tras ciclo verificando si hay algo que hacer). Ineficiente si los dispositivos rara vez requieren atención. Menor rendimiento en sistemas multitarea.
   - Uso típico: Sistemas embebidos simples o cuando el dispositivo es muy rápido.

2. Interrupciones

   - Mecanismo: El dispositivo envía una señal al procesador solo cuando necesita atención. El procesador pausa lo que está haciendo para atenderlo.
   - Ventajas: Más eficiente: el procesador trabaja en otras tareas hasta que se le interrumpe. Ahorra recursos y mejora el rendimiento. Ideal para sistemas con múltiples tareas o dispositivos.
   - Desventajas: Es más complejo de implementar. Requiere controladores de interrupciones (hardware y software). Si hay muchas interrupciones, puede saturar el sistema (interrupciones anidadas o tormentas de interrupciones). Overhead: Guardar/restaurar el contexto consume ciclos de CPU.
   - Uso típico: Sistemas multitarea (ej: SO's modernos) o dispositivos lentos (teclado, disco).

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

1. Mayor eficiencia del procesador: El procesador no pierde tiempo revisando constantemente si un dispositivo necesita atención. Solo actúa cuando realmente se necesita, liberando tiempo de procesamiento para otras tareas.

2. Mejor aprovechamiento del tiempo de CPU: Mientras no ocurren interrupciones, el procesador puede ejecutar otras instrucciones útiles o tareas concurrentes.

3. Menor consumo de energía: Al no estar en bucles de verificación constantes, el procesador puede entrar en estados de bajo consumo (idle o sleep) hasta que se le interrumpa.

4. Mayor capacidad de respuesta: El sistema puede reaccionar casi de inmediato cuando ocurre un evento externo importante, ya que las interrupciones tienen alta prioridad.

5. Permite multitarea real: La CPU puede alternar entre múltiples procesos o hilos mientras los dispositivos trabajan en segundo plano.

6. Escalabilidad: Se pueden gestionar múltiples dispositivos sin necesidad de aumentar la carga del procesador proporcionalmente. Es más escalable que el sondeo, donde agregar más dispositivos aumentaría el tiempo perdido revisando cada uno.

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

En MIPS32, la gestión de interrupciones se realiza mediante un conjunto de registros especiales que forman parte del Coprocesador 0 (CP0). Estos registros permiten configurar, manejar y obtener información sobre las interrupciones y excepciones. Los principales registros son:

1. Status (Registro de Estado - $12): Controla la habilitación de interrupciones. Contiene bits como IE (Interrupt Enable) y IM (Interrupt Mask).
2. Cause (Registro de Causa - $13): Indica la causa de la excepción o interrupción. Contiene bits como IP (Interrupt Pending) y el código de la excepción.
3. EPC (Exception Program Counter - $14): Guarda la dirección de la instrucción que fue interrumpida, para poder retomarla luego del manejo.
4. BadVAddr (Bad Virtual Address - $8): Guarda la dirección que causó una excepción de memoria (por ejemplo, acceso a dirección no válida).
5. Compare ($11): Usado junto con el registro Count para generar interrupciones de temporizador. No implementado en MARS.
6. Count ($9): Incrementa continuamente y se compara con Compare para generar interrupciones periódicas. No implementado en MARS.
7. Config ($16): Configuración general del sistema, como el tamaño de la memoria caché y el modo de endianness. No implementado en MARS.

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

Cuando ocurre una interrupción, el procesador detiene temporalmente la ejecución del programa que estaba corriendo para atender un evento externo. Este cambio de flujo requiere que se guarde el contexto del programa interrumpido, es decir, toda la información que permite reanudar su ejecución exactamente donde se dejó. El contexto incluye el valor del contador de programa (PC), los registros que se estén usando, así como el estado general del procesador.

Guardar el contexto es esencial porque, durante la ejecución de la rutina de servicio (ISR), el procesador podría usar los mismos registros que el programa principal estaba utilizando. Si no se guardan esos valores, se perdería

información y, al volver de la interrupción, el programa podría continuar con datos incorrectos, causando errores o comportamientos inesperados.

Además, conservar el contexto permite que, una vez que finaliza la atención de la interrupción, el procesador pueda restaurar el estado anterior y continuar con el programa como si nada hubiera pasado. Esto es especialmente importante en sistemas multitarea o de tiempo real, donde múltiples eventos pueden ocurrir y ser gestionados sin afectar la lógica del programa principal.

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

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

En MIPS32, una excepción es un evento que interrumpe el flujo normal de ejecución debido a una condición anómala. Estas son cuatro situaciones comunes en las que se puede generar una excepción:

1. Desbordamiento aritmético: Ocurre cuando el resultado de una operación aritmética (como una suma) excede el rango que puede representar el registro. Por ejemplo, al sumar dos enteros positivos que superan el límite máximo de 32 bits.

2. Fallo de dirección de memoria (Address Error): Se produce cuando se accede a una dirección de memoria no alineada (por ejemplo, acceder a una palabra (4 bytes) en una dirección que no es múltiplo de 4) o a una dirección inválida.

3. Instrucción no válida (Reserved Instruction): Se genera si el procesador encuentra una instrucción que no reconoce o que no está permitida en modo de usuario.

4. Llamada al sistema (Syscall): Es una excepción controlada y voluntaria. Se usa para invocar servicios del sistema operativo desde un programa de usuario.

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

Las excepciones pueden generarse en varias de estas etapas, dependiendo del tipo de error. Aquí las más relevantes:

- IF - Instruction Fetch: Fallo de página (TLB Miss): La dirección de la instrucción no está en la TLB. Acceso a memoria inválida: Dirección

fuera del espacio de memoria válido. Esto pasa por que la CPU no puede leer la instrucción solicitada.

- ID – Instruction Decode: Puede generarse una excepción si se detecta una instrucción no válida (por ejemplo, una instrucción reservada o mal codificada). También se puede producir aquí una excepción por syscall.
- EX – Execute: En esta etapa se ejecutan las operaciones aritméticas. Por tanto, puede generarse un desbordamiento aritmético o una división por cero (si se maneja como excepción). MIPS solo lanza una excepción por desbordamiento en operaciones con signo (como add), no con addu.
- MEM – Memory Access: Aquí pueden ocurrir errores de acceso a memoria, como fallos de dirección (por ejemplo, una carga mal alineada), o violaciones de acceso si se intenta leer/escribir en una dirección inválida.
- WB – Write Back: Aunque esta etapa normalmente no genera excepciones, en ciertos sistemas puede usarse para comprobar restricciones de escritura (como protección de memoria), aunque no es común en implementaciones básicas de MIPS.

7 Estrategias de tratamiento de excepciones e interrupciones

7.1 Explica las diferencias entre interrupciones y excepciones (¿son síncronas o asíncronas?)

En un sistema MIPS32, tanto interrupciones como excepciones provocan un desvío del flujo normal del programa hacia una rutina especial de manejo (rutina de servicio). Sin embargo, existen diferencias clave entre ambas:

1. Interrupción
    - Origen: Eventos externos al programa (hardware). Ej: teclado, timer.
    - Asíncronas: Ocurren en cualquier momento, independientemente de la instrucción ejecutada.
    - Control: Generadas por dispositivos externos.
    - Ejemplos: Timer, Teclado, Disco duro.
    - ¿Se puede ignorar?: Algunas son enmascarables (ej: IRQ).
2. Excepción
    - Origen: Eventos sincrónicos al programa (software/hardware). Ej: overflow, acceso inválido. Siempre ocurren en el mismo punto del programa si se repite la ejecución.

- Síncronas: Ocurren durante la ejecución de una instrucción específica.
- Control: Generadas por la CPU al detectar un error o condición especial.
- Ejemplos: División por cero, Instrucción inválida, Fallo de TLB.
- ¿Se puede ignorar?: No ignorables (ej: excepciones críticas).

7.2 Describe brevemente dos estrategias para tratar excepciones en un sistema MIPS32 ¿Cómo se redirige la ejecución hacia la rutina de servicio? ¿Cuál es la función del registro EPC (Exception Program Counter)?

En MIPS32, cuando ocurre una excepción, el sistema necesita redirigir el flujo normal de ejecución hacia una rutina de servicio que se encargue de manejarla. Para ello, se utilizan dos estrategias principales.

La primera estrategia consiste en redirigir todas las excepciones hacia una dirección fija de memoria, que por defecto es 0x80000180. Esta dirección actúa como punto de entrada a una rutina general encargada de atender cualquier tipo de excepción. Desde allí, el sistema examina el contenido del registro Cause, que indica qué tipo de excepción ocurrió (por ejemplo, una instrucción no válida, un desbordamiento aritmético, o una llamada al sistema). Este enfoque es simple, pero obliga a que el código en esa dirección tenga una lógica capaz de identificar y manejar múltiples tipos de excepciones.

La segunda estrategia es el uso de vectores de excepción personalizados, también conocidos como vectores especializados. En esta modalidad, cada tipo de excepción puede redirigirse automáticamente a una dirección distinta en memoria, lo que permite que cada una tenga su propia rutina específica. Esta opción es más eficiente en sistemas complejos, ya que evita tener una única rutina que verifique el tipo de excepción. Para habilitar este comportamiento, es necesario configurar ciertos bits en el registro Status, como el bit BEV (Bootstrap Exception Vector), y definir correctamente las ubicaciones de memoria para cada tipo de excepción.

En ambas estrategias, un paso fundamental es el uso del registro EPC (Exception Program Counter). Este registro guarda la dirección de la instrucción que estaba en ejecución cuando ocurrió la excepción, permitiendo que, una vez resuelto el problema, el procesador pueda volver a ejecutar desde ese punto mediante la instrucción especial eret (exception return). Sin este mecanismo, sería imposible continuar el programa correctamente después de una interrupción inesperada.

8 Habilitación de interrupciones en dispositivos y procesador

8.1 Explica cómo se habilitan las interrupciones en: El teclado, La pantalla, El procesador (detalla qué bits del registro Status deben modificarse).

1. **Teclado**: En MIPS, el teclado se representa como un dispositivo de entrada mapeado en memoria o controlado por una línea de interrupción en el controlador de E/S. Para habilitar interrupciones del teclado, el sistema operativo o programa debe escribir en el registro de control del dispositivo de teclado, activando su bit de interrupción. 0xFFFF0000: Registro de control (habilita interrupciones). 0xFFFF0004: Registro de datos (almacena la tecla presionada).

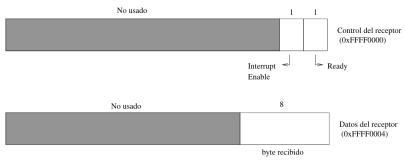

Figure 2: MMIO del teclado

2. **La pantalla**: La pantalla se trata como un dispositivo de salida. Para habilitar sus interrupciones, también se debe configurar su registro de control (o registro de interrupción), permitiendo que la pantalla (o su controlador) genere una interrupción al completar una operación, como mostrar un bloque de datos. 0xFFFF0008: Registro de control (interrupciones). 0xFFFF000C: Registro de datos (carácter a imprimir).

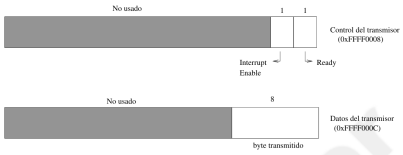

Figure 3: MMIO de la pantalla

3. **El procesador (registro Status)**: En MIPS32, para que el procesador acepte interrupciones, deben activarse los bits adecuados del registro Status (registro 12 de Coprocesador 0). Específicamente: Bit IE (bit 0): es el bit Interrupt Enable global. Debe estar en 1 para permitir interrupciones. Bits IM[7:0] (bits 15 a 8): son los Interrupt Mask bits, que permiten habilitar o bloquear interrupciones individuales. Bit EXL (bit 1): es el bit Exception Level. Cuando está en 1, las interrupciones están deshabilitadas porque el sistema ya está atendiendo una excepción.

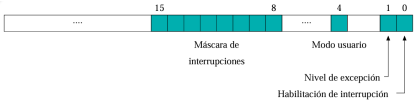

Figure 4: Registro Status

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

Si los dispositivos como el teclado o la pantalla tienen habilitadas sus interrupciones, pero el procesador no tiene habilitado el bit IE ni el correspondiente bit IM, entonces aunque el dispositivo intente interrumpir al procesador, la señal será ignorada. El procesador no interrumpirá su ejecución normal, y la rutina de servicio no será ejecutada.

Esto implica que el dispositivo puede estar generando señales de interrupción correctamente, pero como el procesador no las está aceptando, el sistema no reaccionará ante esos eventos. En consecuencia, pueden perderse datos (por ejemplo, pulsaciones del teclado) o dejarse tareas sin completar (como mostrar información en pantalla), dependiendo del diseño del sistema.

Por eso, es necesario que tanto el dispositivo como el procesador estén coordinados: los dos deben tener las interrupciones habilitadas para que el mecanismo funcione correctamente.

9 Procesamiento de interrupciones

9.1 Describe paso a paso qué ocurre cuando se produce una interrupción de reloj: Desde que el evento ocurre hasta que la rutina de servicio termina. ¿Qué registros se guardan y restauran? ¿Qué hace el hardware y qué hace el software (sistema operativo o rutina)?

1. Ocurre el evento (interrupción de reloj): El timer (reloj) alcanza un valor predefinido (ej: contador llega a cero) y envía una señal de interrupción (IRQ) a la CPU.

2. El hardware del procesador detecta la interrupción: El procesador verifica periódicamente si hay interrupciones pendientes y si están habilitadas en el registro Status (bit IE – Interrupt Enable y los bits IM[7:0] – Interrupt Mask).

3. Se detiene la ejecución normal: Si las interrupciones están habilitadas, el procesador detiene la instrucción que está ejecutando (la deja terminar si ya inició) y cambia el flujo de ejecución.

4. Guarda el contexto del proceso actual: El procesador guarda en el registro EPC (Exception Program Counter) la dirección de la siguiente instrucción a ejecutar (o de la actual si fue interrumpida antes de comenzar). Guarda el valor del registro Status para poder restaurarlo más adelante.). También se guardan los registros generales usados por el

programa que estaba en ejecución (esto lo hace el sistema operativo o la rutina de interrupción).

5. Cambia el modo de ejecución a modo Kernel (supervisor): Se deshabilitan temporalmente otras interrupciones (modificando el bit IE) y se transfiere el control a la rutina de servicio de interrupción (ISR) ubicada en una dirección fija como 0x80000180 (en MIPS32).

6. La rutina de servicio maneja la interrupción: El software (generalmente parte del sistema operativo) ejecuta acciones como: Actualizar el reloj del sistema. Verificar si se deben despertar procesos que estaban dormidos. Posiblemente iniciar un cambio de contexto si hay un proceso con mayor prioridad.

7. Restaurar el contexto: La rutina guarda cualquier resultado necesario y luego restaura los registros generales, el Status y el EPC para que el proceso pueda continuar como si nada hubiera pasado.

8. Se retorna de la interrupción: Se ejecuta la instrucción eret (Exception Return) que: Restaura el registro Status. Usa el valor del EPC para reanudar la ejecución donde se dejó.

¿Qué registros se guardan y restauran?

- Por hardware: EPC (para saber dónde continuar después de la interrupción). Parte del Status.
- Por software (rutina o sistema operativo): Registros generales ($t0-$t9, $s0-$s7, etc.) que estén en uso. Status completo. Otros registros especiales si son necesarios (por ejemplo, HI/LO si se usa multiplicación/división).

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

Guardar el contexto es crítico para evitar que el programa interrumpido pierda información o se comporte incorrectamente después de la interrupción. Si no se guarda el contexto: Las variables del proceso podrían corromperse. Se podría volver a ejecutar desde una posición errónea (si EPC no se guarda). Se podrían habilitar otras interrupciones prematuramente (si Status no se guarda).

10 Interrupciones de reloj y control de ejecución

10.1 Explica cómo una interrupción de reloj puede usarse para: Evitar que un programa quede en un bucle infinito. Finalizar programas que superan un tiempo máximo de ejecución.

Evitar que un programa quede en un bucle infinito:
Las interrupciones de reloj se generan periódicamente por el temporizador del sistema (ej: cada 10 ms). El sistema operativo puede usar estas interrupciones para recuperar el control del procesador incluso si un programa entra en un bucle infinito. Al generarse una interrupción de reloj, se ejecuta una rutina de servicio que puede verificar cuánto tiempo ha estado ejecutándose un programa. Si el programa ha usado demasiado tiempo de CPU sin ceder el control (por ejemplo, por estar en un bucle), el sistema puede decidir finalizarlo o suspenderlo y dar paso a otro proceso (esto se llama planificación o scheduling). Esto es crucial para mantener la estabilidad del sistema y evitar que un solo proceso acapare el procesador.

Finalizar programas que superan un tiempo máximo de ejecución:
El sistema operativo puede asociar un temporizador con cada proceso, configurándolo para un tiempo máximo de ejecución permitido. Cuando se inicia un programa, se establece este temporizador, y si el programa no finaliza antes de que el tiempo expire, se genera una interrupción. La rutina correspondiente podrá entonces tomar acciones como terminar el proceso, lanzar una advertencia, o registrar el evento. Este mecanismo se utiliza, por ejemplo, para evitar que procesos maliciosos o defectuosos bloqueen el sistema.

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

Si el programa finaliza antes de que ocurra la interrupción de reloj, el software (es decir, el sistema operativo) debe cancelar o reajustar el temporizador asociado a ese proceso. Esto es importante para evitar interrupciones innecesarias que generen carga adicional en el sistema. Además, al liberar correctamente los recursos asignados (como los registros de tiempo o los bloques de control del proceso), se mejora la eficiencia general del sistema y se previenen errores relacionados con temporizadores desincronizados.

11 Análisis y Discusión de los Resultados

Los ejercicios implementados en MIPS32 para el simulador MARS cumplen con los objetivos planteados: el buffer circular de teclado y el semáforo controlado por pulsador demuestran un manejo básico de entrada/salida y temporización mediante syscalls. Sin embargo, ambos programas presentan limitaciones significativas debido a su dependencia del polling, una estrategia que consume ciclos de CPU de forma innecesaria y limita la eficiencia del sistema.

En el caso del buffer de teclado, aunque el algoritmo de buffer circular funciona correctamente, la ausencia de interrupciones reales obliga a verificar constantemente el estado del teclado y el temporizador, lo que resulta en un uso ineficiente de recursos. Una implementación con interrupciones hardware (como las disponibles en MARS para el teclado y el timer) permitiría liberar la CPU para otras tareas mientras espera eventos externos.

Por otro lado, el semáforo simula adecuadamente las transiciones entre estados, pero su diseño bloqueante impide que el sistema realice otras operaciones durante los periodos de espera. Aquí, las interrupciones de timer podrían mejorar drásticamente el diseño, permitiendo que el semáforo opere en segundo plano mientras se ejecutan otras rutinas.