Universidad de Carabobo
Facultad experimental de ciencias y tecnología
(FacYT)
Bárbula-Carabobo

Informe
Practica de Laboratorio 3

Bachiller:
Samuel Cordero 31.678.592
Dervis Martínez 31.456.326

Docente:
José Canache

Bárbula, 29 de julio del 2025

1. Explique cómo se organiza la memoria cuando un sistema utiliza memory-mapped I/O. ¿En qué región de memoria se suelen mapear los dispositivos? ¿Qué implicaciones tiene para las instrucciones lw y sw?

Cuando un sistema utiliza Memory-Mapped I/O (E/S mapeada en memoria), la organización de la memoria se caracteriza por unificar el espacio de direcciones de la memoria principal (RAM, ROM) y el de los dispositivos periféricos. En lugar de tener puertos de E/S separados o instrucciones especiales para la comunicación con los dispositivos, los registros de control, estado y datos de los periféricos se asignan a direcciones específicas dentro del mismo espacio de direcciones que utiliza la CPU para acceder a la memoria convencional. Esto significa que, desde la perspectiva del procesador, un registro de un dispositivo periférico se comporta y se accede como si fuera una celda de memoria más. No hay una distinción fundamental a nivel de la unidad de procesamiento entre acceder a un byte de datos en la RAM o leer el estado de un puerto en un chip de E/S; ambos son tratados como operaciones de acceso a una dirección de memoria.

¿En qué región de memoria se suelen mapear los dispositivos?

Los dispositivos de E/S mapeados en memoria generalmente se asignan a una región específica y predefinida del espacio de direcciones de memoria que no se superpone con la memoria principal (RAM o ROM). Esta región suele estar en la parte más alta del espacio de direcciones del procesador. Por ejemplo, en un sistema de 32 bits donde las direcciones van de 0x00000000 a 0xFFFFFFFF:

- La **memoria RAM** a menudo ocupa las direcciones más bajas (ej., 0x00000000 a 0x0FFFFFFF).
- La **ROM o la memoria flash** del sistema podría estar en una región intermedia o al inicio.
- Los **dispositivos periféricos y sus registros de E/S** se suelen mapear en las direcciones más altas (ej., de 0x10000000 o 0xFFFF0000 hacia arriba, o en rangos específicos dentro de la jerarquía de buses como PCI).

La elección de mapearlos en la parte alta se debe a varias razones:

- Evita conflictos con la RAM y la ROM, que suelen ubicarse en las direcciones bajas.
- Permite un crecimiento más flexible de la memoria principal sin invadir el espacio de E/S.
- Simplifica la decodificación de direcciones, ya que rangos de direcciones distintos se pueden manejar con lógica de hardware más sencilla para seleccionar el chip adecuado.

¿Qué implicaciones tiene para las instrucciones lw y sw?

La principal y más significativa implicación del memory-mapped I/O para las instrucciones `lw` (load word - cargar palabra) y `sw` (store word - almacenar palabra) es que estas mismas instrucciones se utilizan tanto para acceder a la memoria principal como para comunicarse con los dispositivos de E/S.

- **lw (Load Word):** Cuando la CPU ejecuta una instrucción `lw` con una dirección que corresponde a un registro de un dispositivo periférico, el procesador lee el valor presente en ese registro del dispositivo y lo carga en uno de sus registros internos. Esto permite al programa leer el estado de un dispositivo (ej., si está ocupado o listo), o leer datos de entrada (ej., la temperatura de un sensor, una pulsación de tecla).
- **sw (Store Word):** De manera similar, cuando la CPU ejecuta una instrucción `sw` con una dirección que corresponde a un registro de un dispositivo, el valor de un registro interno del procesador se escribe en ese registro del dispositivo. Esto permite al programa enviar comandos al dispositivo (ej., iniciar una operación, encender un componente), o enviar datos de salida (ej., datos a una impresora, a una pantalla).

Implicaciones clave:

- **Uniformidad del Espacio de Direcciones:** Elimina la necesidad de un conjunto separado de instrucciones de E/S (como IN y OUT en arquitecturas de puerto-mapeado, como x86). Esto simplifica el diseño del conjunto de instrucciones del procesador.
- **Uso de Instrucciones Estándar:** Las operaciones de E/S se realizan utilizando las mismas instrucciones de carga y almacenamiento que se usan para acceder a la memoria RAM. Esto significa que todas las características del procesador que se aplican a las operaciones de memoria (como los modos de direccionamiento, la caché, las protecciones de memoria, las interrupciones) también pueden aplicarse a las operaciones de E/S.
- **Menos Complejidad del Compilador/Programador:** Los programadores no tienen que aprender y usar instrucciones especiales para la E/S; pueden tratar los dispositivos como extensiones de la memoria, simplificando la programación de bajo nivel y la creación de controladores de dispositivos.
- **Consideraciones de Rendimiento y Efectos Secundarios:**
  - **Caché:** Si los registros de los dispositivos se almacenan en caché, podría haber problemas de coherencia, ya que el estado del dispositivo puede cambiar externamente sin que la caché se actualice. Por esta razón, las regiones de memoria mapeadas a E/S a menudo se configuran como **no cacheadas** (uncached) para garantizar que cada acceso lea o escriba directamente en el dispositivo.
  - **Efectos Secundarios:** Las lecturas o escrituras en registros de E/S pueden tener .efectos secundarios"(por ejemplo, una lectura puede limpiar un `buffer`, o una escritura puede iniciar una operación de hardware), lo cual no es el caso con la memoria RAM normal. Los programadores deben ser conscientes de estos efectos.

2. ¿Cuál es la principal diferencia entre memory-mapped I/O y la entrada/salida por puertos? ¿Qué ventajas y desventajas tiene cada enfoque? ¿Por qué MIPS32 utiliza principalmente memory-mapped I/O?

La principal diferencia entre Memory-Mapped I/O (E/S mapeada en memoria) y la Entrada/Salida por puertos (Port-Mapped I/O o Isolated I/O) radica en cómo el procesador interactúa con los dispositivos periféricos a nivel de hardware y software.

Memory-Mapped I/O (E/S Mapeada en Memoria):

- **Concepto:** Los registros de control, estado y datos de los dispositivos de E/S se asignan a direcciones específicas dentro del mismo espacio de direcciones de memoria que utiliza la CPU para acceder a la RAM y la ROM. Desde la perspectiva del procesador, un dispositivo periférico se ve y se accede como si fuera una ubicación de memoria más.
- **Acceso:** Se utilizan las mismas instrucciones de carga (`lw`, `lb`, etc.) y almacenamiento (`sw`, `sb`, etc.) que se usan para la memoria convencional.

Entrada/Salida por Puertos (Port-Mapped I/O / Isolated I/O):

- **Concepto:** Los dispositivos de E/S tienen un espacio de direcciones separado y distinto del espacio de direcciones de la memoria principal. Se accede a ellos a través de un bus de E/S independiente o con señales de control diferentes.
- **Acceso:** Se requieren instrucciones especiales y dedicadas para la E/S (como `IN` y `OUT` en la arquitectura x86), que el procesador utiliza para especificar un número de puerto en lugar de una dirección de memoria.

Ventajas y Desventajas de cada Enfoque:

Memory-Mapped I/O:

Ventajas:

- **Simplificación del Conjunto de Instrucciones del Procesador:** No se necesitan instrucciones de E/S especiales; se reutilizan las instrucciones existentes para memoria (`lw`, `sw`). Esto reduce la complejidad del diseño de la CPU.
- **Flexibilidad de Modos de Direccionamiento:** Todos los modos de direccionamiento complejos disponibles para la memoria (indexado, indirecto, etc.) pueden aplicarse directamente a los registros de los dispositivos de E/S, lo que permite un acceso más flexible y potente.
- **Consistencia en el Manejo de Memoria/E/S:** Los mismos mecanismos de protección de memoria, gestión de interrupciones y, potencialmente, la caché (aunque las regiones de E/S se marcan como no cacheadas para coherencia) se aplican de forma uniforme a todo el espacio de direcciones.

- **Mayor Velocidad Potencial:** En algunos casos, si los buses de memoria son más rápidos que un bus de E/S dedicado, el acceso a E/S mapeada en memoria puede ser más rápido, ya que utiliza el bus principal.

Desventajas:

- **Consumo del Espacio de Direcciones:** Los dispositivos de E/S çonsumen"parte del espacio de direcciones de memoria disponible, lo que puede limitar la cantidad máxima de RAM que se puede instalar en el sistema, especialmente en arquitecturas con un espacio de direcciones limitado (aunque esto es menos un problema en sistemas de 32 o 64 bits con grandes espacios de direcciones).
- **Posibles Problemas de Coherencia de Caché:** Si los registros de los dispositivos se almacenan en caché, esto podría llevar a que la CPU opere con datos desactualizados si el hardware del dispositivo cambia el registro directamente. Por ello, las regiones MMIO suelen configurarse como no cacheadas, lo que puede impactar ligeramente el rendimiento en comparación con el acceso a la RAM cacheadas.
- **Efectos Secundarios Implícitos:** Las operaciones de lectura o escritura en registros de dispositivos pueden tener efectos secundarios (ej., una lectura puede borrar un `buffer`), lo que requiere que los programadores sean conscientes de ello.

Entrada/Salida por Puertos (Port-Mapped I/O):

Ventajas:

- **Espacio de Direcciones Separado:** La memoria RAM tiene su propio espacio de direcciones completo, sin ser reducida"por los dispositivos de E/S. Esto es ventajoso en arquitecturas con espacios de direcciones pequeños (ej., microcontroladores de 8 bits).
- **Clara Distinción Lógica:** El uso de instrucciones de E/S dedicadas (`IN`/`OUT`) proporciona una clara distinción semántica entre las operaciones de memoria y las operaciones de E/S, lo que puede facilitar la depuración y el entendimiento a nivel de software.
- **Manejo Específico de E/S:** Permite que el diseño del procesador maneje las operaciones de E/S con un conjunto de señales de control y arbitraje de bus distinto, lo que puede ser útil para ciertos tipos de periféricos o para implementar diferentes políticas de acceso.

Desventajas:

- **Complejidad del Conjunto de Instrucciones del Procesador:** Requiere la inclusión de instrucciones de E/S adicionales (`IN`, `OUT`), lo que aumenta la complejidad del diseño de la unidad de control del procesador.
- **Menos Flexibilidad en Modos de Direccionamiento:** Las instrucciones de E/S dedicadas suelen tener modos de direccionamiento más limitados (a menudo solo acceso directo o indirecto a través de un registro específico) en comparación con las operaciones de memoria.
- **Duplicidad de Lógica:** Las funciones de control de memoria y E/S deben replicarse en cierta medida (arbitraje de bus, señales de habilitación, etc.), lo que puede llevar a un diseño de hardware más complejo.

¿Por qué MIPS32 utiliza principalmente Memory-Mapped I/O?

MIPS32 (y muchas otras arquitecturas RISC modernas como ARM) utiliza principalmente Memory-Mapped I/O por las siguientes razones clave, que son intrínsecas a los principios de diseño RISC:

- **Filosofía RISC (Reduced Instruction Set Computer):** El objetivo principal de las arquitecturas RISC es simplificar el conjunto de instrucciones del procesador para permitir un `pipelining` más eficiente y una ejecución más rápida. Al usar `lw` y `sw` para todo, MIPS evita la necesidad de instrucciones especializadas para E/S, lo que encaja perfectamente con esta filosofía. Un conjunto de instrucciones más pequeño y uniforme es más fácil de diseñar, optimizar y depurar.
- **Rendimiento y Optimización del Pipeline:** Tener un conjunto de instrucciones reducido y ortogonal (donde las operaciones no tienen efectos secundarios inesperados en otras operaciones) permite que las etapas del `pipeline` del procesador sean más simples y predecibles. Las instrucciones `lw` y `sw` ya están diseñadas para interactuar con la jerarquía de memoria, y extender su uso a los periféricos aprovecha esta infraestructura existente sin añadir ramificaciones complejas al `pipeline`.
- **Grandes Espacios de Direcciones:** Las arquitecturas MIPS32 (32 bits) y MIPS64 (64 bits) disponen de espacios de direcciones muy grandes (2^{32} y 2^{64} bytes, respectivamente). Esto significa que la "pérdida"de algunas direcciones de memoria para mapear dispositivos de E/S es insignificante y no limita prácticamente la cantidad de memoria RAM utilizable. Por lo tanto, las desventajas de consumo de espacio de direcciones del MMIO son mínimas para MIPS.

3. En un sistema con memory-mapped I/O: ¿Qué problemas pueden surgir si dos dispositivos usan direcciones solapadas? ¿Cómo se evita este conflicto?

En un sistema donde la entrada/salida (E/S) se gestiona mediante el mapeo de memoria (memory-mapped I/O), un problema significativo que puede surgir es el **solapamiento de direcciones**. Esto ocurre cuando dos o más dispositivos de hardware intentan utilizar la misma dirección de memoria o rangos de direcciones que se superponen.

Cuando las direcciones se solapan, el procesador puede experimentar ambigüedad. Si la CPU intenta leer o escribir en una dirección que está asignada a dos dispositivos diferentes, no sabrá a cuál de ellos acceder. Esto puede llevar a:

- **Comportamiento impredecible:** El procesador podría interactuar con el dispositivo incorrecto, resultando en lecturas de datos erróneas, escrituras de datos corruptas, o incluso la ejecución de comandos no intencionados en el dispositivo equivocado.
- **Corrupción de datos:** La información destinada a un dispositivo podría ser escrita en otro, sobrescribiendo sus registros o datos internos, lo que puede causar fallos del sistema o inestabilidad.

- **Conflictos de control:** Si ambos dispositivos tienen registros de control en la misma dirección, un intento de configurar uno podría inadvertidamente configurar o desconfigurar el otro, llevando a un estado inoperable.
- **Bloqueos del sistema:** En casos severos, los conflictos de E/S pueden provocar que el sistema se congele o se reinicie inesperadamente.

Para evitar este conflicto de solapamiento de direcciones, se utilizan varias estrategias, principalmente durante la fase de diseño del hardware y la configuración del sistema operativo:

- **Planificación del Mapa de Memoria (Memory Map Planning):** Es fundamental asignar rangos de direcciones de memoria únicos y no superpuestos a cada dispositivo de hardware desde el inicio del diseño del sistema. Los ingenieros y arquitectos de sistemas crean un mapa de memoria detallado que especifica qué rango de direcciones está reservado para la memoria RAM, la ROM, y cada periférico conectado.
- **Decodificación de Direcciones (Address Decoding):** El hardware del sistema (a menudo a través de un controlador de bus o lógica de decodificación de direcciones) se encarga de asegurar que cada dirección de memoria solicitada por la CPU sea dirigida únicamente al dispositivo o componente de memoria correcto. Cuando la CPU coloca una dirección en el bus de direcciones, la lógica de decodificación la analiza y activa solo el chip o dispositivo que corresponde a ese rango específico, ignorando los demás.
- **Uso de Buses y Controladores Específicos:** En sistemas más complejos, se pueden utilizar buses y controladores de E/S especializados (como PCI, PCIe, USB, etc.) que tienen sus propios mecanismos de asignación de direcciones y arbitraje. Estos mecanismos garantizan que cada dispositivo conectado al bus reciba un espacio de direcciones único y que las comunicaciones se dirijan correctamente.
- **Firmware y Software de Configuración:** Durante el arranque del sistema (BIOS/UEFI) y la carga del sistema operativo, se realiza un proceso de detección y configuración de dispositivos que incluye la asignación o verificación de sus direcciones de E/S mapeadas en memoria para evitar conflictos. El sistema operativo mantiene un registro de las asignaciones de direcciones para garantizar que las aplicaciones y los controladores accedan a los dispositivos correctos.

Al implementar estas medidas, se garantiza que cada dispositivo o componente de memoria tenga su propio .[es]pacio.[ex]clusivo en el mapa de direcciones, lo que permite al procesador comunicarse de manera clara y sin ambigüedades con cada uno de ellos.

4. ¿Por qué se considera que el memory-mapped I/O simplifica el diseño del conjunto de instrucciones de un procesador? ¿Qué tipo de instrucciones adicionales serían necesarias si se usara E/S por puertos?

El Memory-Mapped I/O (E/S mapeada en memoria) se considera que simplifica significativamente el diseño del conjunto de instrucciones de un procesador porque unifica

el acceso a la memoria principal y a los dispositivos periféricos bajo un mismo modelo de operación. En este enfoque, los registros de control, estado y datos de los dispositivos de entrada/salida se asignan a direcciones específicas dentro del espacio de direcciones de memoria del procesador.

Esta unificación tiene varias implicaciones que conducen a la simplificación del diseño del procesador:

- **Reutilización de Instrucciones Existentes:** Con el E/S mapeada en memoria, el procesador no necesita un conjunto especial de instrucciones para comunicarse con los periféricos. Las mismas instrucciones estándar de carga (`lw`, `lb`, etc.) y almacenamiento (`sw`, `sb`, etc.) que se utilizan para acceder a la RAM se emplean también para leer y escribir en los registros de los dispositivos. Esto significa que los diseñadores del procesador no tienen que añadir lógica compleja para decodificar y ejecutar un conjunto completamente nuevo de instrucciones dedicadas a la E/S.

- **Arquitectura de Bus Simplificada:** El bus de direcciones, el bus de datos y las líneas de control del procesador pueden diseñarse de manera uniforme para manejar todos los tipos de accesos (a memoria y a E/S). No se requiere una distinción o mecanismos adicionales en el bus para diferenciar entre un acceso a memoria y un acceso a un puerto de E/S.

- **Aprovechamiento de Modos de Direccionamiento:** Todos los modos de direccionamiento avanzados (directo, indirecto, indexado, basado en registro, etc.) que el procesador ya soporta para la memoria pueden ser utilizados también para acceder a los registros de los dispositivos de E/S. Esto ofrece una gran flexibilidad y potencia a los programadores de dispositivos sin requerir hardware adicional en el procesador.

- **Manejo de Interrupciones y Excepciones Consistente:** Los mecanismos de protección de memoria, el manejo de interrupciones y la gestión de la caché (aunque para E/S mapeada en memoria las regiones suelen ser no cacheadas para evitar problemas de coherencia) pueden aplicarse de manera más coherente y uniforme a todo el espacio de direcciones, incluyendo las áreas mapeadas a E/S.

Por otro contrario, si el sistema utilizara E/S por puertos (Port-Mapped I/O o Isolated I/O), el diseño del conjunto de instrucciones del procesador sería más complejo porque requeriría instrucciones adicionales específicas para la E/S. Un ejemplo clásico de esto se ve en la arquitectura x86, que utiliza este enfoque.

Los tipos de instrucciones adicionales que serían necesarias incluyen:

- **Instrucciones de Entrada (IN):** Para leer datos de un puerto de E/S. Estas instrucciones especificarían un número de puerto (en lugar de una dirección de memoria) desde el cual se debía leer el dato.

- **Instrucciones de Salida (OUT):** Para escribir datos en un puerto de E/S. Similarmente, estas instrucciones especificarían un número de puerto al cual se debía enviar el dato.

Estas instrucciones `IN` y `OUT` operarían a través de un bus de E/S separado o líneas de control distintas a las usadas para el acceso a la memoria. El procesador necesitaría lógica adicional para decodificar estas nuevas instrucciones, generar las señales de control

adecuadas para el bus de E/S y gestionar el flujo de datos a través de los puertos de E/S. Esta duplicidad en el manejo de accesos (uno para memoria, otro para E/S) es lo que añade complejidad al diseño del conjunto de instrucciones y del procesador en general.

5. ¿Qué ocurre a nivel del bus de datos y direcciones cuando el procesador accede a una dirección de memoria que corresponde a un dispositivo? ¿Cómo sabe el hardware que debe acceder a un periférico en lugar de la RAM?

Cuando el procesador accede a una dirección de memoria que, en realidad, corresponde a un dispositivo periférico mapeado en memoria, a nivel de los buses de datos y direcciones, la operación es externamente idéntica a un acceso a la memoria RAM convencional. Esto es lo que ocurre:

- **En el Bus de Direcciones:** El procesador coloca la dirección de memoria que desea leer o escribir en el bus de direcciones. Por ejemplo, si quiere leer el estado de un sensor mapeado en la dirección `0x10000004`, esa dirección se transmite por el bus de direcciones.
- **En el Bus de Control:** Simultáneamente, el procesador activa las líneas de control apropiadas en el bus de control. Esto incluye señales que indican si la operación es una lectura (`MEMR#` o similar) o una escritura (`MEMW#` o similar), y la habilitación de la memoria (`MEMEN#` o similar).
- **En el Bus de Datos:** Si es una operación de escritura, el dato que el procesador quiere enviar se coloca en el bus de datos. Si es una operación de lectura, el bus de datos está listo para recibir el dato del dispositivo o la memoria.

¿Cómo sabe el hardware que debe acceder a un periférico en lugar de la RAM?

La clave para que el hardware sepa si debe acceder a un periférico o a la RAM reside en la **lógica de decodificación de direcciones** y los **circuitos de selección de chip (chip select)**, que son componentes críticos del controlador de memoria o del chipset del sistema. Aquí está el proceso:

- **Lógica de Decodificación de Direcciones:** Cuando el procesador coloca una dirección en el bus de direcciones, no solo la RAM está .[es]cuchando.[es]a dirección. Una unidad de hardware dedicada, llamada **decodificador de direcciones**, examina los bits más significativos de la dirección. Este decodificador está diseñado para reconocer los rangos de direcciones específicos asignados a cada componente del sistema:
  - Si los bits de la dirección indican que está dentro del rango asignado a la **RAM**, el decodificador de direcciones activará la señal de selección de chip (`CS#`) para los módulos de RAM.

- Si los bits de la dirección indican que está dentro del rango asignado a un **dispositivo periférico** (como un controlador de E/S, un temporizador, un controlador de interrupciones, etc.), el decodificador activará la señal de selección de chip (`CS#`) correspondiente a ese periférico específico.

- **Señales de Selección de Chip (Chip Select - CS#):** Cada chip de memoria (RAM, ROM) y cada dispositivo periférico en el sistema tiene una o más entradas de "selección de chip". Un chip solo se activa y responde a las operaciones de lectura/escritura en el bus cuando su línea de `CS#` está activa (normalmente en bajo). La lógica de decodificación de direcciones se asegura de que **solo una** señal de `CS#` esté activa para un rango de direcciones dado en un momento específico, evitando conflictos.

En resumen:

- **A nivel de bus, la operación es la misma:** El procesador simplemente pone una dirección y una operación (lectura/escritura) en el bus.

- **La inteligencia está en el decodificador de direcciones:** Este componente ïntercepta"la dirección y, basándose en la configuración del mapa de memoria del sistema, activa la señal de selección de chip del componente correcto (ya sea RAM o un periférico), y desactiva todas las demás.

- **El hardware del componente activado (RAM o periférico) es el que responde:** Solo el dispositivo cuya señal `CS#` está activa responderá a la solicitud del procesador, poniendo los datos en el bus (para una lectura) o aceptando los datos (para una escritura).

6. ¿Es posible que un programa normal (sin privilegios) acceda a un dispositivo mapeado en memoria? ¿Qué mecanismos de protección existen para evitar accesos no autorizados?

En un sistema operativo moderno, la regla general es que **un programa normal (sin privilegios)**, es decir, una aplicación de usuario que no se ejecuta en modo kernel o supervisor, **no puede acceder directamente a un dispositivo mapeado en memoria** ni a ninguna otra ubicación de hardware crítica. Este es un principio fundamental de la seguridad y estabilidad de los sistemas operativos multiproceso.

Permitir el acceso directo a hardware por parte de cualquier programa de usuario sería una receta para el desastre, ya que una aplicación maliciosa o mal programada podría:

- **Corromper el sistema:** Alterar configuraciones críticas de hardware, dañar el sistema de archivos, o incluso causar un bloqueo total del sistema.

- **Comprometer la seguridad:** Acceder a información sensible de otros procesos o del propio sistema (como claves de cifrado o datos privados).

- **Degradar el rendimiento:** Monopolizar recursos de hardware o interferir con el funcionamiento de otros dispositivos.

Mecanismos de Protección para Evitar Accesos No Autorizados:

Para prevenir estos accesos no autorizados y asegurar la estabilidad y seguridad del sistema, se implementan varios mecanismos de protección clave:

- **Modos de Operación de la CPU (Privilege Levels):** Los procesadores modernos, incluyendo MIPS, operan en diferentes modos de privilegio. Típicamente, hay al menos dos modos:
  - **Modo Supervisor (o Kernel/Privilegiado):** Este es el modo con los más altos privilegios, donde se ejecuta el núcleo del sistema operativo (kernel). En este modo, el procesador tiene acceso completo a todas las direcciones de memoria, todos los registros de control de la CPU, y todos los dispositivos de hardware.
  - **Modo Usuario (o No Privilegiado):** En este modo se ejecutan las aplicaciones de usuario. Los programas en modo usuario tienen un acceso restringido; no pueden ejecutar ciertas instrucciones privilegiadas ni acceder directamente a ciertas regiones de memoria (incluyendo aquellas donde están mapeados los dispositivos de E/S).

  Cuando un programa de usuario intenta acceder a una dirección de E/S mapeada en memoria o ejecutar una instrucción privilegiada, el hardware del procesador detecta esta violación de privilegio y genera una **excepción** o **trampa** (trap). El control se transfiere entonces al kernel, que maneja la situación (generalmente terminando el programa infractor o denegando el acceso).

- **Unidad de Gestión de Memoria (MMU - Memory Management Unit):** La MMU es un componente de hardware esencial en la mayoría de los procesadores modernos. Su función principal es traducir las **direcciones lógicas/virtuales** generadas por los programas a **direcciones físicas** reales en la memoria. La MMU utiliza **tablas de páginas** (gestionadas por el kernel) para realizar esta traducción y para aplicar permisos de acceso. Cada página de memoria puede tener permisos asociados (lectura, escritura, ejecución, y, crucialmente, si es accesible desde el modo usuario o solo desde el modo kernel). Las regiones de memoria donde están mapeados los dispositivos de E/S se configuran en las tablas de páginas de la MMU con permisos que **solo permiten el acceso desde el modo kernel**. Si un programa en modo usuario intenta acceder a una de estas direcciones, la MMU detectará una violación de permisos y generará una excepción, impidiendo el acceso directo.

- **Controladores de Dispositivos (Device Drivers):** Cuando un programa de usuario necesita interactuar con un dispositivo de hardware (por ejemplo, leer de un sensor o escribir en una impresora), no lo hace directamente. En su lugar, realiza una **llamada al sistema (syscall)** al kernel del sistema operativo. El kernel, que se ejecuta en modo privilegiado, es el único que puede acceder directamente a los registros de los dispositivos. La solicitud del programa de usuario es procesada por el **controlador de dispositivo** (device driver) correspondiente, que es una pieza de software específica del kernel para ese hardware. El controlador de dispositivo realiza las operaciones necesarias de lectura/escritura en los registros mapeados en memoria del dispositivo en nombre del programa de usuario, y luego le devuelve el resultado. Esto asegura que todas las interacciones con el hardware estén mediadas

y controladas por el kernel, que aplica las políticas de seguridad y gestiona los recursos de forma segura.

En pocas palabras, la combinación de modos de operación de la CPU, la gestión de la memoria por la MMU y el uso obligatorio de controladores de dispositivos a través de llamadas al sistema garantiza que los programas normales no puedan manipular directamente el hardware mapeado en memoria, manteniendo la integridad, seguridad y estabilidad del sistema.

7. ¿Qué técnicas se pueden emplear para evitar esperas activas innecesarias al interactuar con dispositivos?

Cuando se interactúa con dispositivos de hardware, especialmente aquellos que requieren tiempo para completar una operación (como leer un sensor, escribir en un disco, o esperar una entrada del usuario), una técnica simple pero ineficiente es la espera activa (polling). Esta técnica implica que el procesador verifica repetidamente el estado del dispositivo en un bucle cerrado hasta que la operación se completa. El problema es que, durante este tiempo, la CPU está constantemente ocupada y no puede realizar ninguna otra tarea útil, lo que desperdicia ciclos de procesamiento valiosos.

Para evitar estas esperas activas innecesarias, se emplean principalmente dos técnicas:

E/S Dirigida por Interrupciones (Interrupt-Driven I/O):

- **Concepto:** En lugar de que el procesador pregunte constantemente al dispositivo si ha terminado, el dispositivo es el que notifica al procesador cuando tiene algo que decir o cuando una operación ha concluido. Esto se logra a través de las **interrupciones**.
- **Funcionamiento:**
  - El procesador inicia una operación en el dispositivo (ej., comienza a leer datos de un disco duro).
  - En lugar de esperar activamente, el procesador se libera para realizar otras tareas.
  - Cuando el dispositivo ha completado la operación o tiene datos listos, genera una **señal de interrupción** en una de las líneas de interrupción del procesador (o en un controlador de interrupciones programable).
  - El procesador detiene temporalmente la tarea que estaba ejecutando, guarda su contexto y salta a una rutina especial llamada **Manejador de Interrupciones (Interrupt Service Routine - ISR)** asociada con ese dispositivo.
  - El ISR atiende al dispositivo (ej., lee los datos disponibles, comprueba errores) y luego instruye al controlador de interrupciones para que desactive la señal.
  - Finalmente, el procesador restaura su contexto y reanuda la tarea que estaba haciendo antes de la interrupción.

- **Ventajas:** Permite que la CPU realice múltiples tareas de manera eficiente, mejorando el rendimiento general del sistema, ya que el tiempo de CPU no se desperdicia en esperas ociosas. La CPU solo se involucra cuando es estrictamente necesario.
- **Desventajas:** Puede introducir una latencia mínima debido al tiempo necesario para guardar y restaurar el contexto del procesador. Además, si las interrupciones son muy frecuentes, el overhead de llamar y retornar de los ISRs podría convertirse en un problema.

Acceso Directo a Memoria (DMA - Direct Memory Access):

- **Concepto:** El DMA es una técnica que permite a los dispositivos periféricos transferir datos directamente a la memoria principal o desde ella, **sin involucrar a la CPU** en cada byte transferido.
- **Funcionamiento:**
  - El procesador programa un **controlador DMA** (un chip o módulo dedicado) con la información de la transferencia: dirección de origen del dispositivo, dirección de destino en la memoria, tamaño de la transferencia y si es lectura o escritura.
  - El procesador inicia la transferencia y luego puede liberarse para hacer otras cosas.
  - El controlador DMA toma el control del bus de memoria del sistema y gestiona la transferencia de datos entre el dispositivo y la memoria directamente. La CPU puede quedar temporalmente en un estado de pausa mínima (si el DMA robaçiclos de bus) o puede seguir ejecutándose sin interrupciones significativas.
  - Una vez que la transferencia completa, el controlador DMA envía una **interrupción** al procesador para notificarle que la operación ha terminado.
- **Ventajas:** Elimina la carga de la CPU en transferencias de datos a gran escala, lo que es crucial para dispositivos de alta velocidad (como discos duros, tarjetas de red, GPUs). Esto mejora drásticamente el rendimiento del sistema, libera ciclos de CPU para otras tareas y reduce el overhead de interrupciones al mínimo (solo una interrupción al final de una gran transferencia).
- **Desventajas:** Requiere hardware adicional (el controlador DMA). La configuración inicial del DMA por parte de la CPU puede ser más compleja que un simple acceso `lw/sw`.

Combinación de Técnicas:

En la práctica, los sistemas operativos modernos y los controladores de dispositivos a menudo combinan estas técnicas:

- Se utiliza DMA para transferencias de datos grandes y rápidas, ya que reduce la carga de la CPU.

- Se utiliza la E/S dirigida por interrupciones para notificar al procesador cuando el DMA ha completado su tarea, o para eventos de dispositivos más pequeños y menos frecuentes que no justifican el DMA (como una pulsación de tecla).
- El polling solo se usa en situaciones muy específicas y controladas, como durante la secuencia de arranque del sistema, o cuando se está depurando un nuevo dispositivo, o en entornos donde la latencia es extremadamente crítica y no se pueden permitir los gastos de interrupción/DMA.

Al emplear interrupciones y DMA, los sistemas logran una eficiencia mucho mayor en el manejo de E/S, maximizando el uso de la CPU y proporcionando una experiencia de usuario más fluida y receptiva.

8. Análisis y Discusión de los Resultados:

La presente práctica de laboratorio se ha centrado en explorar y comprender la interacción de un procesador MIPS32 con dispositivos de hardware utilizando el paradigma de E/S Mapeada en Memoria (Memory-Mapped I/O - MMIO). A través de ejercicios prácticos de programación en ensamblador MARS y una discusión teórica sobre los fundamentos de este modelo, hemos podido analizar sus implicaciones en el diseño del sistema, la eficiencia operativa y la seguridad.

Análisis de los Ejercicios Prácticos: Implementación de MMIO en MIPS32

Los ejercicios 1 y 2 representaron una oportunidad fundamental para aplicar los conceptos de MMIO de manera directa en el contexto de la arquitectura MIPS32.

Ejercicio 1: El Sensor de Temperatura

Este ejercicio nos introdujo a la simulación de un sensor de temperatura con tres registros clave mapeados en memoria: SensorControl, SensorEstado y SensorDatos. La implementación requirió el uso de instrucciones estándar de carga (`lw`) y almacenamiento (`sw`) para interactuar con estos registros como si fueran ubicaciones de memoria convencionales.

- **InicializarSensor:** La función InicializarSensor ilustró la fase de configuración de un dispositivo. Al escribir el valor `0x2` en `SensorControl`, se simulaba la activación del sensor. Posteriormente, se utilizó una espera activa (polling) para verificar el registro `SensorEstado`. Este bucle (`wait_for_init`) demostró cómo el procesador sondea continuamente el estado del dispositivo hasta que este indica que está listo (cuando `SensorEstado` almacena un 1). Aunque funcional para la simulación, este método resalta una de las principales ineficiencias del polling: el procesador gasta ciclos valiosos en una espera ociosa.
- **LeerTemperatura:** La función LeerTemperatura se encargó de la lectura del dato. Primero, se verificó nuevamente el `SensorEstado` para asegurar que el sensor estuviera listo para la lectura o para detectar un error (-1). Si el estado era 1, el

valor de la temperatura se leía directamente de `SensorDatos` usando `lw`, y se retornaban dos valores (temperatura y código de éxito/error) en los registros `$v0` y `$v1`, respectivamente. Esto mostró la simplicidad del acceso a datos del dispositivo una vez que está listo.

Ejercicio 2: El Módulo de Tensión Arterial

Similar al ejercicio 1, el módulo de tensión arterial implicaba la interacción con registros `TensionControl`, `TensionEstado`, `TensionSistol` y `TensionDiastol`. Este ejercicio consolidó la comprensión de los patrones de interacción de MMIO.

- **controlador_tension:** La función `controlador_tension` replicó el patrón de iniciar una operación mediante una escritura (`sw 1, TensionControl`) y luego esperar (`lw` en un bucle de polling sobre `TensionEstado`) a que el dispositivo completara su tarea. Una vez que `TensionEstado` pasaba a 1, se procedía a leer los dos valores resultantes, `TensionSistol` y `TensionDiastol`, directamente de sus respectivas direcciones mapeadas en memoria, retornándolos también en `$v0` y `$v1`.
- **Confirmación del Patrón MMIO:** Ambos ejercicios demostraron claramente que, en MIPS32, la comunicación con el hardware se realiza mediante las mismas instrucciones de carga y almacenamiento utilizadas para la memoria RAM, validando el principio central de MMIO. La necesidad de una simulación manual en MARS para los cambios de estado y valores de los registros del "sensor"subrayó la diferencia entre la operación de bajo nivel del procesador y la complejidad interna del periférico real.

Discusión de los Conceptos Fundamentales de E/S Mapeada en Memoria

Más allá de la implementación práctica, la práctica abordó conceptos teóricos cruciales para entender el MMIO en profundidad.

- **Organización de la Memoria y Funcionamiento de lw/sw (Pregunta 1):** Se analizó cómo el MMIO unifica el espacio de direcciones de memoria para RAM, ROM y dispositivos de E/S. Los periféricos suelen mapearse en las regiones más altas del espacio de direcciones para evitar colisiones con la memoria principal. La implicación directa para las instrucciones `lw` y `sw` es que estas se convierten en el **único** medio para interactuar con todos los componentes del sistema. No se requieren instrucciones especiales de E/S; un `lw` o un `sw` a una dirección mapeada a un dispositivo se traduce directamente en una operación de E/S. Esta uniformidad simplifica enormemente el diseño del procesador y del compilador.
- **Diferencias entre MMIO y E/S por Puertos (Pregunta 2):** La discusión se extendió a la principal diferencia entre MMIO y la E/S por puertos. Mientras que MMIO integra dispositivos en el espacio de direcciones de memoria, la E/S por puertos mantiene un espacio de direcciones separado para los periféricos, requiriendo instrucciones de E/S dedicadas (`IN/OUT`).
  - **Ventajas de MMIO:** La simplicidad del conjunto de instrucciones (reutilización de `lw/sw`), la flexibilidad de los modos de direccionamiento y la coherencia en el manejo de memoria/E/S son puntos fuertes.

- **Desventajas de MMIO:** El consumo de espacio de direcciones y la necesidad de manejar la coherencia de caché (marcando las regiones de E/S como no cacheadas) son desventajas menores en arquitecturas modernas con grandes espacios de direcciones.
- **Ventajas de E/S por Puertos:** Preserva el espacio de direcciones completo para la RAM y ofrece una distinción lógica clara entre memoria y E/S.
- **Desventajas de E/S por Puertos:** Mayor complejidad del conjunto de instrucciones y menos flexibilidad en los modos de direccionamiento de E/S.

MIPS32, al ser una arquitectura RISC, prioriza la simplicidad y el rendimiento del `pipeline`. El MMIO se alinea perfectamente con esta filosofía, permitiendo un conjunto de instrucciones reducido y eficiente al evitar la adición de instrucciones de E/S especializadas.

- **Manejo de Conflictos por Direcciones Solapadas (Pregunta 3):** Se abordó la problemática crítica de las direcciones solapadas en un sistema MMIO, donde dos dispositivos podrían responder a la misma dirección. Esto puede llevar a lecturas incorrectas, corrupción de datos y fallos del sistema. La solución fundamental radica en una **robusta lógica de decodificación de direcciones** implementada en el hardware del sistema (chipset o controlador de bus). Esta lógica asegura que, al examinar los bits más significativos de la dirección, solo se active la señal de selección de chip (`CS#`) del componente de memoria o del dispositivo periférico correcto, garantizando así un acceso único y sin ambigüedades.

Aspectos de Seguridad y Eficiencia en la Interacción con Dispositivos

Finalmente, se exploraron las consideraciones prácticas de seguridad y eficiencia que van más allá del simple acceso a registros.

- **Acceso de Programas No Privilegiados y Mecanismos de Protección (Pregunta 6):** Se estableció claramente que los programas de usuario (sin privilegios) no pueden ni deben acceder directamente a los dispositivos mapeados en memoria. Esta restricción es vital para la estabilidad y seguridad del sistema operativo. Los mecanismos que hacen cumplir esta protección incluyen:
  - **Modos de Operación de la CPU:** Los procesadores operan en modos privilegiados (kernel) y no privilegiados (usuario), restringiendo el acceso a direcciones críticas.
  - **Unidad de Gestión de Memoria (MMU):** La MMU, mediante tablas de páginas, traduce direcciones virtuales a físicas y aplica permisos de acceso, permitiendo que solo el kernel acceda a las regiones de E/S mapeadas.
  - **Controladores de Dispositivos (Device Drivers):** Los programas de usuario interactúan con el hardware indirectamente a través de llamadas al sistema (`syscalls`), que son procesadas por los drivers del kernel. Esto garantiza que todas las operaciones de hardware estén mediadas y supervisadas por el sistema operativo.

- **Evitando Esperas Activas Innecesarias (Pregunta 7):** El polling utilizado en los ejercicios, aunque sencillo, es ineficiente porque desperdicia ciclos de CPU. Se discutieron dos técnicas superiores para mejorar la eficiencia en la interacción con dispositivos:

  - **E/S Dirigida por Interrupciones:** El dispositivo notifica al procesador (mediante una interrupción) solo cuando ha completado una operación o tiene datos listos. Esto libera a la CPU para realizar otras tareas mientras espera.
  - **Acceso Directo a Memoria (DMA):** Un controlador DMA dedicado permite que los dispositivos transfieran grandes bloques de datos directamente a/desde la memoria principal sin la intervención de la CPU para cada byte. El procesador solo se involucra para iniciar la transferencia y recibe una interrupción al finalizar.

  La combinación de interrupciones y DMA es el estándar en sistemas modernos para optimizar el rendimiento de la E/S y asegurar que la CPU esté disponible para el procesamiento de aplicaciones.

Conclusión

La práctica ha consolidado la comprensión del Memory-Mapped I/O como un pilar fundamental en la arquitectura de computadoras modernas, especialmente en procesadores como MIPS32. Hemos visto cómo simplifica el diseño del conjunto de instrucciones al unificar el acceso a memoria y dispositivos, permitiendo el uso de las versátiles instrucciones `lw` y `sw` para toda interacción. Sin embargo, esta simplicidad a nivel del procesador se complementa con la necesidad de una cuidadosa ingeniería a nivel de sistema: una sólida lógica de decodificación de direcciones para evitar conflictos de hardware, y robustos mecanismos de protección del sistema operativo (modos de privilegio, MMU, controladores) para salvaguardar la integridad y seguridad. Finalmente, para superar las limitaciones de eficiencia del `polling`, las interrupciones y el DMA se revelan como soluciones indispensables para sistemas de alto rendimiento, permitiendo que los dispositivos y la CPU colaboren de manera óptima.