Informe Practica 3

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

29 de Julio 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?

En un sistema con *memory-mapped I/O*, los dispositivos de entrada/salida (E/S) se mapean en el espacio de direcciones de memoria principal. Esto significa que las direcciones de memoria reservadas para E/S se asignan a registros de control, estado o datos de los dispositivos, en lugar de a celdas de memoria RAM. Los dispositivos suelen mapearse en regiones específicas del espacio de direcciones, como las direcciones altas o reservadas por el sistema.

Para las instrucciones `lw` (load word) y `sw` (store word), esto implica que pueden utilizarse para interactuar con los dispositivos de E/S como si fueran memoria. Por ejemplo, al leer (`lw`) una dirección mapeada a un registro de estado, se obtiene el estado del dispositivo, y al escribir (`sw`) en una dirección mapeada a un registro de control, se envía un comando al dispositivo. No se requieren instrucciones especiales para E/S, lo que simplifica el diseño del procesador.

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?

- **Diferencia**:
  - *Memory-mapped I/O* integra los dispositivos en el espacio de direcciones de memoria, usando instrucciones de carga/almacenamiento (`lw/sw`) para E/S.
  - *E/S por puertos* utiliza un espacio de direcciones separado y requiere instrucciones específicas (como `in` y `out` en x86) para acceder a los dispositivos.

E/S Mapeada en Memoria

Ventajas:

- Se pueden utilizar las mismas instrucciones de acceso a memoria para acceder a los registros de E/S.
- No se requieren instrucciones especiales de E/S.
- Se puede utilizar cualquier tipo de direccionamiento disponible para la memoria.
- Se pueden mapear tantos registros como direcciones de memoria haya disponibles.

Desventajas:

- Se reduce el espacio de direcciones disponible para la memoria principal.
- Se requiere un mecanismo para proteger los registros de E/S del acceso no autorizado por parte de programas de usuario (normalmente gestionado por el sistema operativo).

E/S No Mapeada en Memoria (E/S Aislada)

Ventajas:

- No se reduce el espacio de direcciones de la memoria principal.
- Es más fácil controlar el acceso a los registros de E/S, ya que se requieren instrucciones específicas.

Desventajas:

- Se necesitan instrucciones especiales de E/S (como IN y OUT), lo que puede complicar la lógica del procesador.
- El número de registros de E/S que se pueden direccionar suele ser limitado.
- Se pueden requerir señales de control adicionales en el bus del sistema para distinguir entre accesos a memoria y accesos a E/S.

MIPS32 utiliza principalmente *memory-mapped I/O* porque su arquitectura RISC favorece la simplicidad del conjunto de instrucciones. Al no necesitar instrucciones especiales para E/S, se reduce la complejidad del hardware y se mantiene un diseño más limpio y eficiente.

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 con **Memory-Mapped I/O**, los dispositivos de entrada/salida se comunican con el CPU a través de direcciones de memoria específicas. Si dos dispositivos comparten o se solapan en las mismas direcciones, pueden surgir los siguientes problemas:

- **Acceso erróneo a dispositivos**: El CPU podría leer o escribir en una dirección esperando interactuar con un dispositivo, pero en realidad estaría accediendo al otro.
- **Activación no deseada**: Algunos dispositivos pueden reaccionar incluso ante una lectura. Si comparten una dirección, ambos podrían activarse simultáneamente.
- **Corrupción de datos**: Es posible sobrescribir datos importantes si el CPU accede a una dirección compartida sin saberlo.
- **Lecturas peligrosas**: En Memory-Mapped I/O, incluso leer una dirección puede desencadenar una acción en el hardware.

Soluciones

Para evitar estos conflictos, se aplican las siguientes estrategias, también válidas en arquitecturas como MIPS32:

- **Asignación única de direcciones**: Cada dispositivo debe tener una dirección única en el espacio de memoria.
- **Uso de instrucciones byte específicas**: Se utilizan instrucciones como `BITB` o `TSTB` para acceder a un solo byte, evitando tocar direcciones adyacentes.
- **Decodificación precisa en hardware**: El sistema de memoria incluye lógica que redirige correctamente las solicitudes del CPU a cada dispositivo.
- **Separación de registros**: Los registros de estado y datos de los dispositivos se asignan a direcciones distintas, evitando solapamientos.

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* utiliza las mismas instrucciones de carga/almacenamiento (`lw/sw`) para acceder tanto a memoria como a dispositivos de E/S. Esto elimina la necesidad de instrucciones especializadas para E/S.

En cambio, con E/S por puertos, el procesador requeriría instrucciones adicionales como: in para leer de un puerto. out para escribir en un puerto. Esto aumenta la complejidad del diseño del procesador y del compilador, ya que se deben manejar dos espacios de direcciones (memoria y puertos) con instrucciones distintas

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 mapeada a un dispositivo:

1. La dirección se coloca en el bus de direcciones.
2. El controlador de memoria o un circuito de decodificación determina, basándose en el rango de direcciones, si la solicitud corresponde a RAM o a un dispositivo.
3. Si es un dispositivo, el controlador habilita el acceso al periférico correspondiente y bloquea el acceso a RAM para esa dirección.
4. Los datos se transfieren a través del bus de datos entre el procesador y el dispositivo.

El hardware sabe diferenciar entre RAM y periféricos gracias a la decodificación de direcciones, donde ciertos rangos están preasignados para E/S durante el diseño del sistema.

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?

No, un programa normal sin privilegios no puede acceder directamente a un dispositivo mapeado en memoria. Esto se debe a que los sistemas modernos implementan varios mecanismos de protección, tanto a nivel de hardware como de software, para garantizar la seguridad y estabilidad del sistema operativo.

¿Por que?

- Los dispositivos mapeados en memoria (como controladores, periféricos, etc.) están ubicados en **regiones especiales** de la memoria física que el **sistema operativo reserva** para el kernel.
- Un programa normal se ejecuta en **modo usuario**, y este modo tiene restricciones severas de acceso para proteger el sistema contra errores o acciones maliciosas.
- Intentar acceder a esas regiones sin permiso provoca una **excepción de protección** (por ejemplo, una falla de segmentación), lo que generalmente termina el programa.

Mecanismos que impiden el acceso

Mecanismo	Función principal
Modo de ejecución	El procesador distingue entre modo *usuario* y *kernel*. Solo el kernel puede acceder directamente a memoria de dispositivos.
MMU (Memory Management Unit)	Traduce direcciones virtuales a físicas y controla los accesos a cada región.
Tablas de páginas	Determinan qué páginas pueden ser leídas, escritas o ejecutadas por cada proceso.
Syscalls y drivers	El acceso a dispositivos se realiza indirectamente mediante llamadas al sistema y controladores confiables.
Espacio de direcciones	Cada proceso tiene su propio entorno virtual; no puede ver ni modificar el del kernel o de otros procesos.

Esto garantiza que el hardware no pueda ser manipulado directamente por procesos maliciosos o con errores. Así se protege la **integridad del sistema** y se evita que un programa pueda, por ejemplo, modificar el estado de un controlador de disco o leer datos confidenciales desde memoria compartida.

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

Para evitar esperas activas como el *sondeo* (*polling*), que consiste en verificar repetidamente el estado de un dispositivo o registro de hardware para determinar si está listo para realizar una operación, que consumen recursos del CPU, se usan:

- **Interrupciones**: El dispositivo notifica al procesador cuando está listo, liberando al CPU para otras tareas hasta que ocurra la interrupción.
- **DMA (Direct Memory Access)**: Permite a los dispositivos transferir datos directamente a memoria sin intervención del CPU.
- **Buffering**: Almacenar datos en búferes para que el CPU no deba esperar a que el dispositivo esté listo en cada operación.

Estas técnicas mejoran la eficiencia del sistema al reducir el tiempo de espera del CPU.

8. Análisis y Discusión de los Resultados

Aunque la finalidad sea distinta, ambos algoritmos son similares en cuanto a su funcionamiento. Inicializando registros específicos para marcar el correcto funcionamiento del programa. En el sensor de temperatura, al inicializarse el sensor de estado y recibir el valor 1 se inicializa el sensor de control, lo que habilita la lectura de la temperatura y para luego imprimirla con su código. Similar al algoritmo anterior se inicializa el registro de control de tensión para iniciar la medición, se calcula la tensión sistólica y la tensión diastólica y se guardan en sus respectivos registros, y se concluye la medición guardando el valor 1 en el registro de estado. Aunque se utilizan registros particulares, esto permite visualizar en la memoria de datos el estado de los procesos, además de facilitar la utilización de dispositivos de E/S mapeados en memoria.