Informe sobre E/S Mapeada en Memoria Mips Práctica 3

Brigido Noguera (29.525.299)
Ángel Besteiro (30.577.354)

Martes, 29 de Julio de 2025

Informe sobre E/S Mapeada en Memoria

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 E/S mapeada en memoria (Memory-Mapped I/O), los registros de los dispositivos de hardware se tratan como ubicaciones de memoria estándar. Esto significa que, en lugar de tener un espacio de direcciones separado para los puertos de E/S, los registros de los periféricos se asignan a direcciones dentro del espacio de direcciones de memoria principal.

Los dispositivos suelen mapearse en una región de memoria específica, a menudo en la parte superior del espacio de direcciones de memoria, para evitar conflictos con la memoria RAM principal. Sin embargo, la ubicación exacta puede variar dependiendo de la arquitectura y el diseño del sistema.

Para las instrucciones `lw` (load word) y `sw` (store word), esto tiene una implicación significativa: no se necesitan instrucciones especiales para interactuar con los dispositivos de hardware. Las mismas instrucciones que se utilizan para leer (`lw`) y escribir (`sw`) en la memoria RAM se emplean para leer y escribir en los registros de los dispositivos periféricos. Esto simplifica el conjunto de instrucciones del procesador, ya que el hardware puede tratar todas las operaciones de acceso como si fueran accesos a memoria.

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 E/S mapeada en memoria y la E/S por puertos radica en cómo el procesador accede a los dispositivos periféricos.

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

- **Funcionamiento:** Los registros de los dispositivos se asignan a direcciones en el espacio de direcciones de memoria. El procesador utiliza las mismas instrucciones de carga y almacenamiento (`lw`, `sw`) que usa para acceder a la memoria RAM.
- **Ventajas:**
  - **Simplificación del conjunto de instrucciones:** No se requieren instrucciones de E/S especiales, lo que simplifica el diseño del procesador.
  - **Flexibilidad:** Se pueden usar todas las instrucciones de manipulación de datos y direccionamiento disponibles para la memoria también para los dispositivos.

- **Acceso directo:** Los dispositivos pueden ser accedidos directamente por el procesador como si fueran ubicaciones de memoria.

- **Desventajas:**
  - **Espacio de direcciones de memoria consumido:** Parte del espacio de direcciones de memoria se consume por los dispositivos, lo que puede limitar la cantidad de RAM disponible si no se planifica adecuadamente.
  - **Potenciales conflictos de caché:** Los accesos a dispositivos no deberían ser cacheados, lo que requiere mecanismos especiales para deshabilitar el caché para esas regiones.

E/S por Puertos (Port-Mapped I/O / Isolated I/O):

- **Funcionamiento:** Los dispositivos de E/S tienen su propio espacio de direcciones separado del espacio de direcciones de memoria. El procesador utiliza instrucciones de E/S dedicadas (como `IN` y `OUT` en arquitecturas x86) para acceder a estos puertos.

- **Ventajas:**
  - **Espacio de direcciones de memoria no afectado:** El espacio de direcciones de memoria RAM no se ve reducido por los dispositivos de E/S.
  - **Control de acceso:** Las instrucciones de E/S dedicadas pueden ser instrucciones privilegiadas, lo que permite un mejor control de acceso a los dispositivos por parte del sistema operativo.

- **Desventajas:**
  - **Conjunto de instrucciones más complejo:** Requiere instrucciones de E/S adicionales y específicas, lo que aumenta la complejidad del diseño del procesador.
  - **Menos flexibilidad:** Las operaciones disponibles para los puertos pueden ser limitadas en comparación con las operaciones de memoria.

Por qué MIPS32 utiliza principalmente E/S mapeada en memoria: MIPS32 utiliza principalmente E/S mapeada en memoria porque simplifica el diseño del conjunto de instrucciones del procesador. Al tratar los registros de los dispositivos como ubicaciones de memoria, el procesador puede usar las mismas instrucciones de carga y almacenamiento para interactuar tanto con la memoria principal como con los periféricos. Esto reduce la complejidad de la unidad de control del procesador y elimina la necesidad de circuitos de hardware adicionales para manejar un espacio de direcciones de E/S separado.

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 E/S mapeada en memoria, si dos dispositivos usan direcciones solapadas, pueden surgir problemas graves. Cuando el procesador intenta acceder a una dirección

en el espacio de memoria que es utilizada por dos dispositivos diferentes, el resultado es impredecible y puede llevar a:

- **Comportamiento errático:** El procesador podría leer datos de un dispositivo cuando intenta leer del otro, o escribir datos en ambos, lo que causaría un funcionamiento incorrecto.
- **Corrupción de datos:** Las escrituras en una dirección compartida podrían sobrescribir configuraciones o datos críticos de un dispositivo que no era el objetivo.
- **Bloqueos del sistema:** En casos extremos, un conflicto de direcciones podría provocar un bloqueo del sistema o un estado inestable.

Cómo se evita este conflicto: Para evitar conflictos de direcciones, se emplean las siguientes estrategias:

- **Planificación del mapa de memoria:** Durante la fase de diseño del hardware y software, se crea un mapa de memoria detallado que asigna rangos de direcciones únicos a cada dispositivo y a la memoria RAM. Este mapa garantiza que no haya solapamientos.
- **Decodificación de direcciones:** El hardware de decodificación de direcciones es crucial. Cuando el procesador genera una dirección, los circuitos de decodificación determinan a qué componente (RAM, ROM o un periférico específico) corresponde esa dirección y habilitan solo el componente adecuado para la operación de lectura o escritura.
- **Registros de base y límite:** En sistemas más complejos, se pueden usar registros de base y límite (o unidades de gestión de memoria - MMU) para asignar rangos de direcciones a diferentes módulos o procesos, lo que ayuda a evitar conflictos y proporciona protección de memoria.

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 E/S mapeada en memoria simplifica el diseño del conjunto de instrucciones de un procesador porque elimina la necesidad de instrucciones de E/S dedicadas. En cambio, las mismas instrucciones de carga (`lw`) y almacenamiento (`sw`) que se utilizan para acceder a la memoria principal se pueden usar para interactuar con los dispositivos periféricos. Esto significa que el conjunto de instrucciones del procesador es más pequeño y homogéneo, lo que puede conducir a un diseño de hardware más simple y eficiente.

Si se usara E/S por puertos, serían necesarias **instrucciones adicionales** específicas para la entrada/salida. Estas instrucciones podrían incluir:

- `IN` (o similar): Para leer datos de un puerto de E/S específico.

- `OUT` (o similar): Para escribir datos en un puerto de E/S específico.
- Posiblemente, instrucciones para configurar o consultar el estado de los puertos de E/S.

Estas instrucciones adicionales requerirían lógica de decodificación y control separada en la unidad de control del procesador, aumentando su complejidad.

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 corresponde a un dispositivo mapeado en memoria, ocurre lo siguiente a nivel del bus de datos y direcciones:

1. **Generación de Dirección y Control:** El procesador coloca la dirección de memoria deseada en el bus de direcciones y las señales de control (lectura/escritura) en el bus de control.
2. **Decodificación de Dirección:** Un circuito de decodificación de direcciones en el sistema (parte del controlador de memoria o de la lógica del bus) examina la dirección. Este circuito está diseñado para reconocer los rangos de direcciones asignados a la RAM, la ROM y los diversos dispositivos periféricos.
3. **Habilitación del Dispositivo/Memoria:** Basándose en la decodificación de la dirección, el circuito de decodificación habilita el chip de memoria (RAM o ROM) o el controlador del dispositivo periférico correspondiente. Solo el componente habilitado responderá a la transacción del bus.
4. **Transferencia de Datos:**
    - **Lectura:** Si es una operación de lectura, el dispositivo habilitado coloca los datos de su registro (o la memoria, si es una dirección de RAM) en el bus de datos, y el procesador lee esos datos.
    - **Escritura:** Si es una operación de escritura, el procesador coloca los datos en el bus de datos, y el dispositivo habilitado (o la memoria) los recibe y los almacena en su registro (o ubicación de memoria).

¿Cómo sabe el hardware que debe acceder a un periférico en lugar de la RAM? El hardware sabe si debe acceder a un periférico o a la RAM gracias al **circuito de decodificación de direcciones**. Este circuito es una pieza lógica que, dado un rango de direcciones, activa la señal de selección (chip select) de un componente específico (RAM, ROM, o un periférico). Por ejemplo, si el procesador genera una dirección que cae dentro del rango asignado a un sensor de temperatura, el circuito de decodificación activará la línea de selección de chip del sensor y desactivará las de la RAM, asegurando que la operación de lectura/escritura se dirija solo al sensor.

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?

Sí, es posible que un programa normal (sin privilegios) acceda a un dispositivo mapeado en memoria, ya que los registros de los dispositivos se encuentran dentro del espacio de direcciones de memoria accesible. Sin embargo, esto representa un riesgo de seguridad y estabilidad para el sistema.

Mecanismos de protección para evitar accesos no autorizados: Para evitar accesos no autorizados por parte de programas sin privilegios, se emplean varios mecanismos de protección, principalmente a nivel de hardware y sistema operativo:

- **Unidad de Gestión de Memoria (MMU - Memory Management Unit):** La MMU es un componente de hardware que se encuentra entre el procesador y la memoria. Traduce direcciones lógicas (generadas por el procesador) a direcciones físicas y, lo que es más importante, implementa mecanismos de protección:
  - **Protección de páginas:** La MMU divide el espacio de direcciones en páginas y puede asignar permisos (lectura, escritura, ejecución) a cada página. El sistema operativo configura la MMU para que las páginas que contienen los registros de los dispositivos de E/S tengan permisos restringidos, por ejemplo, solo accesibles en modo privilegiado (kernel mode).
  - **Segmentación:** Aunque menos común que la paginación en arquitecturas modernas para la protección general de memoria, la segmentación también puede usarse para aislar regiones de memoria.
- **Modos de operación del procesador (Privilege Levels):** Los procesadores modernos operan en diferentes modos de privilegio (por ejemplo, modo usuario y modo kernel/supervisor). El sistema operativo se ejecuta en modo kernel, que tiene acceso total a todos los recursos de hardware, incluyendo los registros de E/S mapeados en memoria. Los programas de usuario se ejecutan en modo usuario, que tiene un conjunto de permisos restringido. Cualquier intento de un programa de usuario de acceder a una dirección de E/S protegida generará una excepción (fault), que es manejada por el sistema operativo, impidiendo el acceso.
- **Llamadas al sistema (System Calls):** Los programas de usuario no acceden directamente a los dispositivos. En su lugar, solicitan al sistema operativo que realice la operación de E/S en su nombre a través de llamadas al sistema. El sistema operativo (que se ejecuta en modo privilegiado) realiza el acceso al dispositivo y luego devuelve el resultado al programa de usuario. Esto centraliza el control de E/S y permite al sistema operativo aplicar políticas de seguridad.

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

Las esperas activas (busy-waiting o polling) ocurren cuando el procesador verifica repetidamente el estado de un dispositivo en un bucle cerrado, consumiendo ciclos de CPU valiosos que podrían usarse para otras tareas. Para evitar esperas activas innecesarias al interactuar con dispositivos, se emplean las siguientes técnicas:

- **Interrupciones (Interrupts):**
  - **Concepto:** En lugar de que el procesador pregunte continuamente al dispositivo si está listo, el dispositivo notifica al procesador cuando ha completado una operación o cuando hay datos disponibles, generando una señal de interrupción.
  - **Funcionamiento:** Cuando ocurre una interrupción, el procesador suspende su tarea actual, guarda su contexto, y salta a una rutina de servicio de interrupción (ISR - Interrupt Service Routine) específica para ese dispositivo. Una vez que la ISR ha manejado la interrupción, el procesador puede reanudar su tarea original.
  - **Ventajas:** Libera al procesador para realizar otras tareas mientras espera que el dispositivo esté listo, mejorando la eficiencia del sistema.
  - **Ejemplo:** Un sensor de temperatura podría generar una interrupción cuando una nueva lectura esté disponible, o un controlador de disco podría interrumpir al procesador cuando una transferencia de datos ha finalizado.
- **Acceso Directo a Memoria (DMA - Direct Memory Access):**
  - **Concepto:** DMA es un mecanismo que permite a los dispositivos periféricos transferir datos directamente hacia y desde la memoria principal sin la intervención constante del procesador.
  - **Funcionamiento:** El procesador configura un controlador DMA (que es un circuito de hardware dedicado) con la dirección de origen, la dirección de destino y la cantidad de datos a transferir. Una vez configurado, el controlador DMA toma el control del bus del sistema para realizar la transferencia de datos de forma independiente. Una vez que la transferencia ha finalizado, el controlador DMA notifica al procesador mediante una interrupción.
  - **Ventajas:** Reduce significativamente la carga de la CPU en operaciones de E/S intensivas, como la transferencia de grandes bloques de datos desde un disco duro o una tarjeta de red.
- **Sistemas operativos y planificación de tareas:**
  - Los sistemas operativos modernos utilizan planificadores de tareas para alternar entre diferentes procesos. Cuando un proceso necesita esperar por E/S, el sistema operativo puede ponerlo en un estado de "bloqueoz programar otro proceso para que se ejecute. Cuando la E/S se completa (generalmente a través de una interrupción), el sistema operativo puede cambiar el estado del proceso de E/S de nuevo a "listo"para que pueda ser programado para ejecutarse.

Al combinar interrupciones y DMA, el procesador solo necesita intervenir al inicio y al final de una operación de E/S o cuando se produce un evento específico, evitando así las costosas esperas activas.

8. Análisis y Discusión de los Resultados

Este apartado estaría destinado a un análisis de los resultados obtenidos tras la implementación de los ejercicios prácticos propuestos (`InicializarSensor`, `LeerTemperatura`, `controlador_tension`). Se discutiría la correcta funcionalidad de los procedimientos implementados en ensamblador MIPS32, verificando que:

- **InicializarSensor:** Inicie correctamente el sensor de temperatura escribiendo el valor `0x2` en `SensorControl` y espere a que `SensorEstado` indique que el sensor está inicializado (valor 1). Se analizarían los ciclos de reloj y el uso de registros durante la fase de inicialización.
- **LeerTemperatura:** Lea el valor de temperatura de `SensorDatos` una vez que el sensor esté listo. Se validaría el retorno correcto del valor leído y del código de error (-1 en caso de `SensorEstado` = -1, o 0 si la lectura es exitosa). Se discutiría la implementación de la lógica de manejo de errores y cómo se propaga el código de retorno.
- **controlador_tension:** Inicie la medición de tensión arterial escribiendo el valor 1 en `TensionControl`. Se confirmaría que el procedimiento espera activamente (o utiliza una técnica de sondeo) hasta que `TensionEstado` sea 1. Finalmente, se verificaría que los valores de tensión sistólica y diastólica se retornan correctamente en los registros especificados (`$v0` y `$v1`, respectivamente).

Se podrían incluir capturas de pantalla del simulador MARS mostrando la ejecución de los procedimientos, los valores en los registros y en las direcciones de memoria mapeadas a los dispositivos. Además, se discutirían posibles optimizaciones del código, como la reducción de ciclos de reloj, el uso eficiente de los registros y la robustez del manejo de errores. En caso de haber utilizado esperas activas, se justificaría su uso en el contexto de la práctica y se contrastaría con las técnicas de interrupciones o DMA para escenarios de producción. La discusión también podría abordar cualquier desafío encontrado durante la implementación y cómo se resolvió.