PRÁCTICA DE LABORATORIO 3: INFORME

Jose Contin V29.947.026

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?

Cuando un sistema usa memory-mapped I/O (entrada/salida mapeada en memoria), los dispositivos como teclados, pantallas o puertos de comunicación no tienen un espacio propio, sino que "aparecen" en la memoria principal. Es como si, en lugar de tener que usar órdenes especiales para hablar con ellos, el procesador pudiera acceder a ellos simplemente leyendo o escribiendo en ciertas direcciones de memoria, en lugar de utilizar un espacio de direccionamiento separado como en el caso del I/O mapeado por puertos. Esta organización permite que el CPU acceda a los dispositivos mediante las mismas instrucciones de carga/almacenamiento (lw y sw en MIPS) que se usan para operar con la memoria RAM.

En arquitecturas como MIPS32 (y en entornos como MARS), los dispositivos suelen mapearse en zonas reservadas del espacio de direcciones, típicamente en la parte alta de la memoria (por ejemplo, a partir de 0xFFFF0000). Esta región suele estar asociada al kernel, ya que el acceso directo a hardware normalmente requiere privilegios. Por ejemplo:

1. El teclado en MARS aparece en 0xFFFF0004.
2. La consola de salida (Display Data Terminal) usa 0xFFFF000C para enviar caracteres.

Implicaciones para lw y sw
Las instrucciones lw (cargar palabra) y sw (almacenar palabra) pueden leer/escribir tanto en memoria RAM como en registros de dispositivos. (Por ejemplo: lw $t0, 0xFFFF0000($zero) Lee el registro de status de un dispositivo). Esto simplifica el ISA al no requerir instrucciones especializadas

para E/S (como in/out en x86).

Al acceder a direcciones mapeadas, una operación de lectura (lw) podría no solo retornar un valor, sino también modificar el estado del dispositivo (ej: limpiar un flag de interrupción). Una escritura (sw) puede desencadenar acciones físicas (ej: enviar un carácter a un puerto serial).

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 está en cómo el CPU accede a los dispositivos de hardware:

1. Memory-Mapped I/O (MMIO):
    - Los dispositivos se mapean en el espacio de memoria principal.
    - Se usan las mismas instrucciones de carga/almacenamiento (lw, sw, etc.) para interactuar con ellos.
    - Ejemplo en MIPS: Leer de 0xFFFF0000 podría ser consultar el estado del teclado.
2. Entrada/Salida por Puertos (Port-Mapped I/O, PMIO):
    - Los dispositivos tienen un espacio de direccionamiento separado (no comparten memoria con la RAM).
    - Se necesitan instrucciones especiales (in, out, como en x86) para acceder a ellos.
    - Ejemplo en x86: in al, 0x60 lee un byte del puerto del teclado.

Cada método tiene sus pros y contras. El MMIO simplifica el diseño del procesador al no requerir instrucciones especiales, lo que encaja bien con la filosofía RISC de MIPS, donde se prioriza un conjunto reducido de operaciones universales. Además, permite tratar los dispositivos como estructuras de datos en memoria, facilitando la programación. Sin embargo, al compartir el bus de memoria, puede generar cuellos de botella si hay mucha actividad simultánea de RAM y E/S. (Acceso lento si el bus de memoria está saturado.)

El PMIO, por su parte, evita saturar el bus de memoria al usar vías separadas para E/S, lo que puede mejorar el rendimiento en sistemas con operaciones intensivas de dispositivos (como servidores o máquinas industriales). No obstante, exige hardware más complejo (como buses adicionales) y obliga a

manejar instrucciones específicas, algo que arquitecturas como MIPS evitan deliberadamente para mantener su simplicidad.

¿Por qué MIPS32 usa principalmente MMIO?

1. Simplicidad del ISA: MIPS es una arquitectura load/store (solo lw/sw acceden a memoria), y agregar instrucciones especiales para E/S complicaría el diseño.a el diseño del procesador (no necesita instrucciones in/out adicionales).

2. Facilidad de programación: Al usar direcciones de memoria, no hay que aprender un nuevo conjunto de instrucciones para dispositivos.

3. Adaptabilidad en sistemas embebidos: Muchos sistemas MIPS (como routers o consolas antiguas) usan MMIO porque simplifica la conexión de periféricos sin necesidad de hardware adicional para manejo de puertos.

4. Compatibilidad con MARS: El simulador MARS está diseñado para enseñanza, y el MMIO hace más intuitivo el manejo de dispositivos (ejemplo: el teclado y la pantalla se ven como "memoria especial").

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 (MMIO), si dos dispositivos comparten direcciones de memoria solapadas, se producen conflictos críticos que pueden llevar a comportamientos impredecibles. pueden surgir varios problemas:

1. Interferencia en la comunicación: Si un programa intenta escribir en una dirección asignada a dos dispositivos (ej: 0xFFFF0000), ambos recibirán la misma señal, causando comportamientos no deseados. Por ejemplo, enviar un carácter a una pantalla podría accidentalmente activar también un controlador de sonido.

2. Lecturas corruptas: Al leer (lw) una dirección compartida, el CPU podría recibir datos mezclados de ambos dispositivos, generando información errónea.

3. Fallos en el control de hardware: Algunos dispositivos borran flags o registros al leerlos. Si otro dispositivo está mapeado en la misma dirección, se perderían estados importantes.

Cómo se evitan estos conflictos
Para evitar estos conflictos, los sistemas emplean varias estrategias. Durante el diseño del hardware, se asigna a cada dispositivo un rango de direcciones exclusivo, evitando superposiciones. En plataformas como MIPS32 con MARS, estas asignaciones están predefinidas (por ejemplo, el teclado en 0xFFFF0000 y la pantalla en 0xFFFF0008). A nivel de circuito, un decodificador de direcciones garantiza que cada acceso active únicamente el periférico correcto.

En sistemas más complejos, el sistema operativo también interviene, protegiendo las zonas de memoria vinculadas a dispositivos para que aplicaciones de usuario no las modifiquen accidentalmente. Sin estas precauciones, errores de direccionamiento podrían inutilizar componentes esenciales del sistema.

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 simplifica el diseño del conjunto de instrucciones (ISA) porque elimina la necesidad de instrucciones especializadas para E/S. En lugar de requerir operaciones exclusivas para interactuar con dispositivos, el sistema trata los periféricos como posiciones de memoria más. Esto permite que las instrucciones básicas de carga/almacenamiento (lw, sw en MIPS) sean suficientes para manejar tanto la memoria principal como los dispositivos.

Entres sus Ventajas se encuentran: No hay distinción entre acceder a RAM o a un dispositivo: ambos usan lw/sw. Esto reduce la cantidad total de instrucciones necesarias en el procesador. Además El circuito de decodificación de direcciones ya existe para la memoria, por lo que reutilizarlo para E/S evita añadir rutas de control adicionales para puertos separados. Po último, Los desarrolladores no necesitan aprender un nuevo conjunto de instrucciones para E/S (como in/out en x86). Basta con conocer las direcciones mapeadas de los dispositivos.

¿Qué instrucciones adicionales requeriría el enfoque de E/S por puertos?

Si el sistema usara E/S por puertos (como en x86), el procesador necesitaría:

- Instrucciones dedicadas: in (leer de un puerto) y out (escribir en un puerto), con formatos distintos a lw/sw. (Ejemplo: in al, 0x60; out $t1, 0x80)

- Circuitos adicionales: Un bus separado para puertos de E/S y lógica de control específica para manejar estas instrucciones.
- Extensiones en el ISA: Modos de direccionamiento especiales para puertos (ej: puertos de 8, 16 o 32 bits).

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 mapeada a un dispositivo (ej: 0xFFFF0000 en MIPS), el hardware detecta que esa dirección no pertenece a la RAM convencional mediante un decodificador de direcciones. Este circuito, situado entre la CPU y los componentes del sistema, compara la dirección solicitada con rangos predefinidos para periféricos. Si coincide, redirige la operación al bus de E/S correspondiente y activa señales de control específicas (como MEMR/MEMW para RAM vs IOR/IOW para dispositivos, aunque en MMIO se usan las mismas señales que para memoria).

El dispositivo respondiente (ej: un controlador de teclado) reacciona entonces a la lectura/escritura, mientras que la RAM ignora la transacción. La clave está en que el mapa de memoria físico está diseñado para que ciertas zonas "hagan shadowing" a registros de hardware en lugar de a celdas de RAM.

En sistemas simples como MARS, esto se simula mediante tablas internas que asocian direcciones a funciones de E/S. En hardware real, lo gestiona el controlador de memoria (northbridge) o lógica similar.

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 sistemas con protección de memoria (como MIPS32), un programa sin privilegios (user mode) no puede acceder directamente a dispositivos mapeados en memoria. Esto se evita mediante varios mecanismos de protección:

1. Protección por modo de ejecución (el más fundamental): Los procesadores modernos (incluyendo MIPS32) tienen al menos dos modos: kernel mode (privilegiado) y user mode (no privilegiado). Las instrucciones que acceden a zonas MMIO solo se permiten en kernel mode. Al intentar acceder desde user mode, se genera una excepción (generalmente "Segmentation Fault" o "Bus Error").
2. Mapeo de memoria privilegiada: El Memory Management Unit (MMU) marca las páginas que contienen dispositivos como "solo kernel". Cualquier acceso desde user mode es bloqueado por hardware.
3. Protección del sistema operativo: El kernel asigna las direcciones MMIO solo a espacio de kernel durante el boot. Los drivers actúan como intermediarios obligatorios.

Excepciones:
En sistemas embebidos simples sin protección, los programas sí pueden acceder directamente. Algunos SO permiten acceso controlado mediante llamadas al sistema (como mmap en Linux para mapear dispositivos en espacio de usuario).

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

Cuando interactuamos con dispositivos mediante memory-mapped I/O, las esperas activas (polling) son ineficientes porque consumen ciclos de CPU revisando constantemente el estado del dispositivo. Estas son las principales alternativas:

- Interrupciones (la solución más eficiente): El dispositivo notifica al procesador cuando está listo mediante una señal de interrupción. La CPU puede ejecutar otras tareas mientras espera. Ejemplo: En MIPS, el teclado genera una interrupción cuando se presiona una tecla
- DMA (Acceso Directo a Memoria): Un controlador especial transfiere datos directamente entre dispositivo y memoria. El CPU solo programa la transferencia y es notificado al finalizar. Ideal para transferencias grandes (ej: discos duros, tarjetas de red)
- Buffering doble (para dispositivos de E/S secuencial): Se usan dos buffers alternados. Mientras la CPU escribe en un buffer, el dispositivo lee del otro. Elimina esperas en dispositivos de visualización o audio.
- Temporizadores programables: Para dispositivos con tiempos de respuesta predecibles. La CPU programa un delay conocido en lugar de hacer polling.

En MIPS con MMIO, las interrupciones son especialmente relevantes. El coprocesador CP0 maneja las interrupciones, y dispositivos como el teclado en MARS pueden configurarse para generar interrupciones cuando hay datos disponibles, evitando que la CPU tenga que consultar continuamente el registro de estado.

8 Análisis y Discusión de los Resultados

El análisis de ambos ejercicios muestra un patrón común en la interacción con periféricos mediante memoria mapeada en MIPS32. En ambos casos se usa un esquema de tres registros: control (para iniciar operaciones), estado (para verificar disponibilidad) y datos (para los resultados).

En el sensor de temperatura se requiere una inicialización explícita (0x2 en SensorControl) y se manejan múltiples estados (0, 1, -1), lo que añade complejidad en la verificación. El módulo de tensión arterial simplifica este esquema con un único disparador (1 en TensionControl) y solo dos estados (0/1).

Un desafío clave en MARS es la simulación del cambio de estado en los registros, que en hardware real sería automático pero en el simulador requiere intervención manual o suposiciones. Ambos ejercicios usan espera activa (polling), método simple pero ineficiente que consumiría todos los ciclos de CPU en sistemas reales hasta completar la operación.

Estos ejemplos enseñan los fundamentos de E/S mapeada en memoria: protocolos estandarizados, manejo de registros y la importancia de entender el timing hardware-software. La principal limitación es que asumen operaciones atómicas - en entornos más complejos se necesitarían mecanismos adicionales como interrupciones.