Práctica 3: Mapeo de memoria E/S

Rolando Suarez V-30445947
Jesus Muñoz V-27188137

1 Organización de la memoria con Memory-Mapped I/O

Cuando un sistema utiliza **memory-mapped I/O (MMIO)**, los registros de los dispositivos de entrada/salida (E/S) se asignan a direcciones dentro del espacio de direcciones de la memoria principal. Esto significa que el procesador interactúa con los dispositivos de E/S como si estuviera leyendo o escribiendo en ubicaciones de memoria normales.

En la arquitectura **MIPS32**, la memoria principal se organiza en varias secciones:

- **Segmento para texto**: Comienza en la dirección `0x00400000`.
- **Segmento de datos**: Comienza en la dirección `0x10010000`.
- **Heap**: Memoria dinámica que crece hacia arriba para almacenamiento de datos.
- **Stack**: Memoria para registros y variables de ejecución, crece hacia abajo.
- **Segmento de texto del kernel**: Comienza en `0x80000000`.
- **Segmento de datos del kernel**: Comienza en `0x90000000`.

Los registros de **Memory-Mapped I/O** para los dispositivos de E/S se mapean típicamente en una región de memoria específica. En el simulador MARS, esta región comienza físicamente en la dirección `0xFFFF0000`. Esta es la región donde se mapean los dispositivos.

Las instrucciones `lw` (load word) y `sw` (store word) se utilizan como funciones de entrada y salida para comunicarse con los dispositivos, con la memoria como intermediario. La instrucción `lw` sirve como función de lectura; por ejemplo, para leer si una tecla del teclado ha sido presionada. La instrucción `sw` sirve para lo contrario, es decir, para escribir datos en un dispositivo, como enviar un carácter a una pantalla.

2 Diferencia entre Memory-Mapped I/O y E/S por puertos

La principal diferencia entre **Memory-Mapped I/O (MMIO)** y la **entrada/salida por puertos (PMIO)** radica en el espacio de direcciones utilizado para los dispositivos de E/S.

2.1 Ventajas y Desventajas de MMIO

Ventajas de MMIO:

- **No requiere instrucciones especiales**: Utiliza las mismas instrucciones de carga (`lw`) y almacenamiento (`sw`) para acceder a los datos de los dispositivos, lo que simplifica el conjunto de instrucciones del procesador.
- **Permite usar punteros y estructuras**: Facilita el acceso a los dispositivos utilizando las técnicas de programación de memoria estándar.
- **Facilita la integración con compiladores y lenguajes de alto nivel**: Al tratar los dispositivos como ubicaciones de memoria, es un método más flexible y portable.

Desventajas de MMIO:

- **Reduce el espacio disponible en RAM**: Parte del espacio de direcciones de memoria se designa para los dispositivos de E/S, lo que puede limitar la cantidad de RAM disponible para programas.
- **Puede requerir protección adicional**: Es necesario implementar mecanismos de protección para evitar accesos accidentales o no autorizados a los registros de los dispositivos.

2.2 Ventajas y Desventajas de PMIO

Ventajas de PMIO:

- **No ocupa la memoria física**: Utiliza un espacio de direcciones separado y dedicado para la entrada y salida, lo que deja todo el espacio de direcciones de memoria principal disponible para la RAM.
- **Puede simplificar la protección de memoria**: Al tener un espacio de direcciones distinto, la protección de los dispositivos puede ser más directa.

Desventajas de PMIO:

- **Requiere instrucciones especiales**: Necesita instrucciones específicas como `IN` y `OUT` para interactuar con los puertos de E/S.
- **Menos intuitivo para programadores de alto nivel**: La interacción con los dispositivos puede ser menos directa ya que no se utiliza el modelo de memoria estándar.

2.3 Razones del uso de MMIO en MIPS32

MIPS32, al igual que otras arquitecturas RISC, utiliza principalmente **memory-mapped I/O** por varias razones clave:

- **Simplicidad del conjunto de instrucciones**: Evita la necesidad de añadir instrucciones especiales para las tareas de entrada y salida, manteniendo el conjunto de instrucciones lo más pequeño y eficiente posible.
- **Facilidad de programación**: Permite usar las mismas instrucciones, herramientas y técnicas para el acceso a memoria y a los dispositivos, lo que simplifica el desarrollo de software.
- **Diseño limpio y uniforme**: Facilita la estandarización y uniformidad en el uso de dispositivos de entrada y salida, contribuyendo a un diseño de sistema más coherente.

3 Problemas de direcciones solapadas en MMIO y cómo evitarlos

En un sistema con **memory-mapped I/O**, es crucial que cada dispositivo de entrada y salida posea una dirección de memoria única asociada. Si existe un **solapamiento** de direcciones entre dos o más dispositivos, surgen graves problemas de funcionamiento:

- **Ambigüedad del procesador**: El procesador no podría distinguir a qué dispositivo está accediendo, lo que llevaría a comportamientos impredecibles. Por ejemplo, podría intentar leer datos de una pantalla cuando espera la entrada de un teclado.
- **Inestabilidad y falta de fiabilidad**: El sistema en su conjunto se vuelve inestable y poco fiable.
- **Corrupción de datos**: Pueden ocurrir problemas de corrupción de datos si el sistema detecta inconsistencias, ya que los datos podrían provenir de dispositivos distintos.

Para evitar este conflicto, se deben emplear las siguientes estrategias:

- **Asignación de direcciones fija y documentada**: Cada tipo de dispositivo debe tener un rango de direcciones de memoria exclusivo, previamente definido y bien documentado.
- **Mapeo por hardware**: Componentes como los controladores de bus se encargan de asegurar que cada dirección se enrute al dispositivo correcto, gestionando el acceso de manera ordenada.

4 Simplificación del diseño del conjunto de instrucciones con MMIO

El **memory-mapped I/O (MMIO)** simplifica el diseño del conjunto de instrucciones de un procesador porque **no se requiere añadir nuevas instrucciones** para la comunicación con los dispositivos de entrada y salida. Toda la comunicación se realiza mediante las instrucciones ya existentes para cargar (`lw`, `lb`, `lh`, etc.) y almacenar (`sw`, `sb`, `sh`, etc.) datos. Esto resulta en:

- **Uniformidad del diseño**: El procesador no necesita lógica adicional para decodificar y ejecutar instrucciones de E/S específicas.
- **Portabilidad**: Al utilizar instrucciones estándar, el código de E/S es más portable entre diferentes implementaciones de la misma arquitectura.
- **Compatibilidad con la programación de alto nivel**: Los lenguajes de alto nivel pueden acceder a los dispositivos de E/S utilizando punteros y estructuras, de la misma manera que acceden a la memoria RAM.

Si se utilizara **E/S por puertos**, serían necesarias **instrucciones adicionales** como `IN` y `OUT`. Estas instrucciones tendrían que incluir:

- La **identificación del dispositivo** o puerto al que se quiere acceder.
- El **comando** o tipo de operación (lectura o escritura) que se desea realizar.

Esto añade complejidad al conjunto de instrucciones y requiere una lógica adicional en el hardware del procesador para decodificar y ejecutar estas instrucciones especiales. Los sistemas CISC, como los procesadores x86 de Intel y AMD, a menudo emplean este tipo de esquema.

5 Acceso a dispositivos mapeados en memoria a nivel de bus

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:

- El procesador coloca la **dirección del registro del dispositivo** en el bus de direcciones.
- También coloca las **señales de control** apropiadas (lectura/escritura) en el bus de control.
- En el caso de una escritura, los **datos a escribir** se colocan en el bus de datos.

El hardware sabe que debe acceder a un periférico en lugar de a la RAM porque:

- **Decodificación de direcciones**: Los circuitos de hardware de decodificación de direcciones en el sistema están diseñados para reconocer que ciertas rangos de direcciones no corresponden a la RAM física, sino a los registros de los dispositivos de E/S.
- **Espacio reservado**: Como se mencionó, existe un espacio físico de la memoria reservado específicamente para el mapeo de dispositivos (e.g., `0xFFFF0000` en MARS). Cuando una dirección cae dentro de este rango, el controlador de bus o la lógica de interconexión del sistema enruta la solicitud al controlador del dispositivo correspondiente en lugar de a los chips de RAM.
- **Controladores de dispositivos**: Cada dispositivo de E/S tiene un controlador específico que está conectado al bus. Este controlador responde únicamente a las direcciones mapeadas a su dispositivo, interpretando los accesos de memoria como comandos o lecturas/escrituras de sus registros internos.

En esencia, el dato que se está "guardando en memoria" es el comando o la información dirigida al dispositivo.

6 Acceso de programas normales a dispositivos mapeados en memoria

Generalmente, **no es posible** que un programa normal (sin privilegios) acceda directamente a un dispositivo mapeado en memoria. Esto se debe a mecanismos de protección implementados por el sistema operativo y el hardware:

- **Niveles de privilegio**: En la presencia de un sistema operativo, existen diferentes niveles de privilegio. Típicamente, el **nivel kernel** (o modo privilegiado) es el único que tiene acceso directo al espacio de direcciones reservado para el mapeado de E/S. Los programas de usuario (nivel usuario o modo no privilegiado) no tienen permiso para acceder a estas direcciones.
- **Sistema de conversión/traducción de direcciones**: Las unidades de manejo de memoria (MMU) en el procesador, bajo el control del sistema operativo, gestionan la traducción de direcciones virtuales a físicas. Estas MMU pueden configurarse para denegar el acceso a ciertas regiones de memoria (como las de E/S mapeadas) si el acceso proviene de un programa sin los privilegios adecuados.

En simuladores como MARS, estas protecciones a nivel de sistema operativo no están presentes, permitiendo un acceso más directo para fines de aprendizaje. Sin embargo, en un sistema real, la seguridad y estabilidad dependen de estas barreras de protección.

7 Técnicas para evitar esperas activas innecesarias

Al interactuar con dispositivos de entrada y salida, el procesador a menudo necesita esperar a que un dispositivo complete una operación o tenga datos listos. La **espera activa** (o *polling*) implica que el procesador consulta repetidamente el estado del dispositivo, lo cual consume recursos de manera innecesaria, ya que el procesador generalmente opera a una velocidad mucho mayor que el dispositivo.

Para evitar estas esperas activas innecesarias, se pueden emplear las siguientes técnicas:

- **E/S dirigida por interrupciones**: Esta es la técnica más común y eficiente. En lugar de que el procesador consulte al dispositivo, el dispositivo genera una **interrupción** cuando necesita atención del procesador (e.g., una operación ha finalizado, hay datos disponibles, o ha ocurrido un error). El procesador, al recibir la interrupción, suspende su tarea actual y ejecuta una rutina de servicio de interrupción (ISR) para atender al dispositivo. Esto permite que el procesador realice otras tareas mientras el dispositivo está ocupado, y solo se interrumpe cuando es estrictamente necesario.
- **Acceso Directo a Memoria (DMA)**: Para transferencias de datos grandes, la DMA permite que los dispositivos de E/S transfieran datos directamente hacia y desde la memoria principal sin la intervención constante del procesador. El procesador solo inicia la transferencia y es notificado mediante una interrupción cuando la transferencia DMA ha finalizado. Esto libera al procesador de la tarea de mover datos byte a byte o palabra a palabra.

8 Análisis y Discusión de los Resultados

En la implementación de los algoritmos de sensor de temperatura y sensor de tensión, pudimos observar el funcionamiento del **mapeado de memoria para el control de dispositivos de entrada y salida**, al menos de forma simulada. Se creó un apartado de memoria con los nombres de los "comandos" que se deseaban asignar al dispositivo. Esto incluyó:

- Un **comando de control** para encender el dispositivo o prepararlo para la lectura.
- **Comandos de lectura** para obtener el valor de retorno que se supone que el dispositivo debería entregar.

En nuestro caso, estos datos se generaron de manera aleatoria, ya que no se contó con un medidor de tensión o termómetro real conectado al equipo.

Esta práctica también demostró la utilidad de los scripts en MIPS32 destinados a generar números aleatorios dentro de un rango.

Nuestro algoritmo de temperatura fue capaz de:

- Determinar si la lectura de temperatura fue correcta o si hubo un error.
- Mostrar en pantalla la temperatura leída.

Nuestro algoritmo de tensión mostró por pantalla:

- Un mensaje indicando que la medición se estaba realizando en ese momento.
- Los resultados de la medición de tensión sistólica y diastólica.

Estos resultados, aunque simulados, ilustran eficazmente cómo **memory-mapped I/O** permite la interacción del software con el hardware a través de operaciones de memoria estándar.