Informe de Práctica de Laboratorio 3: Memory-Mapped I/O

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

a) Organización de memoria

En *memory-mapped I/O*, el espacio de direcciones de memoria se divide en dos regiones:

- **Memoria principal (RAM/ROM):** Almacena código y datos de aplicaciones (direcciones bajas: 0x00000000)
- **Región de E/S:** Asignada a registros de dispositivos periféricos (direcciones altas)

La CPU accede a ambos espacios usando las mismas instrucciones de memoria.

b) Región de mapeo

Los dispositivos se mapean típicamente en la región alta del espacio de direcciones (ej: 0xFFFF0000 en MIPS32). Esto evita conflictos con la RAM y facilita la expansión de memoria.

c) Implicaciones para lw/sw

- `lw`: Lee registros de dispositivos (estado/datos)
- `sw`: Escribe en registros de control
- **Efectos secundarios:** Accesos pueden desencadenar operaciones físicas (iniciar mediciones, resetear dispositivos)
- **Sincronización:** Requieren verificación de registros de estado

2. Diferencias entre Memory-Mapped I/O y E/S por Puertos

Diferencia principal

- **MMIO:** Espacio unificado (memoria + dispositivos)
- **E/S por puertos:** Espacios separados (instrucciones especiales IN/OUT)

Ventajas/desventajas

Criterio	Memory-Mapped I/O	E/S por Puertos
Espacio direcciones	Reduce espacio disponible	Conserva espacio completo
Instrucciones	Reutiliza lw/sw	Requiere IN/OUT
Programación	Mayor flexibilidad	Modos direccionamiento limitados
DMA	Implementación directa	Compleja

Uso en MIPS32

MIPS32 utiliza principalmente MMIO por:

- Filosofía RISC (instrucciones mínimas)
- Arquitectura load/store
- Simplicidad en diseño de compiladores

3. Conflictos por direcciones solapadas

Problemas

- **Corrupción de datos:** Múltiples dispositivos responden simultáneamente
- **Escrituras accidentales:** Comandos inválidos a dispositivos
- **Comportamiento indefinido:** Fallos aleatorios difíciles de depurar

Prevención

1. **Diseño hardware:** Decodificación estricta de direcciones
2. **Asignación jerárquica:** BIOS/UEFI gestiona direcciones en arranque
3. **Protección SO:** Mapeo solo en espacio kernel
4. **Estándares PCI/PCIe:** Asignación dinámica de direcciones

4. Simplificación del conjunto de instrucciones

Simplificación

MMIO elimina la necesidad de instrucciones especializadas de E/S, utilizando las mismas `lw/sw` para:

- Acceso a memoria
- Control de dispositivos
- Transferencia de datos

Instrucciones necesarias con E/S por puertos

Se requerirían:

- `IN` (lectura de puertos)
- `OUT` (escritura a puertos)
- Instrucciones para manejar buses de E/S
- Canales DMA especializados

5. Acceso a dispositivos a nivel de bus

Proceso

1. CPU coloca dirección en bus (ej: 0xFFFF0010)
2. Decodificador detecta rango de E/S
3. Activa señal *chip select* del dispositivo
4. Desactiva módulos de memoria
5. Dispositivo responde en bus de datos

Identificación periférico

El hardware diferencia mediante:

- Circuitos decodificadores de dirección
- Lógica combinacional específica
- Señales de habilitación por rango

6. Acceso de programas sin privilegios

Restricciones

- **Imposible** en sistemas modernos
- Protección por:
  1. Modos CPU (usuario vs kernel)
  2. MMU (no mapea E/S en espacio usuario)
  3. Bit de privilegio en tablas de páginas

Mecanismos protección

- Fallo de página por violación de privilegio
- Interrupciones de seguridad (SMM)
- Capas de hipervisores

7. Técnicas para evitar esperas activas

- **Interrupciones (IRQ):**
  - Dispositivo notifica cuando está listo
  - CPU ejecuta ISR (Interrupt Service Routine)
- **DMA (Acceso Directo a Memoria):**
  - Transferencias directas dispositivo-memoria
  - CPU sólo configura parámetros
- **Blqueo/planificación:**
  - SO suspende proceso hasta que esté disponible
  - Reasignación de CPU a otros procesos

8. Análisis y Discusión de Resultados

El estudio de *memory-mapped I/O* revela aspectos fundamentales de sistemas embebidos y arquitecturas RISC:

- La integración hardware/software mediante MMIO permite diseños más eficientes en recursos limitados
- La protección de memoria es crítica para estabilidad del sistema
- Las técnicas de sincronización (polling vs interrupciones) presentan compromisos entre latencia y utilización de CPU
- MIPS32 ejemplifica cómo arquitecturas load/store simplifican operaciones complejas

Los ejercicios prácticos demuestran que:

- La abstracción de dispositivos como memoria simplifica programación
- La gestión adecuada de registros evita condiciones de carrera
- El diseño de drivers requiere comprensión profunda de efectos secundarios