1- El mapa de la memoria

Con un bus de direcciones de 32 bits, el espacio de direcciones RISC-V abarca 32 bytes (4 GB) -*direcciones lógicas*-. Las direcciones de palabra son múltiplos de 4 y varían de 0 a 0xFFFFFFFC. La figura 1 muestra un ejemplo de mapa de memoria. Nuestro mapa de memoria divide el espacio de direcciones en cinco partes o segmentos: el texto, los datos globales, los segmentos de datos dinámicos, y los segmentos para los controladores de excepciones y el sistema operativo (SO), que incluye la memoria dedicada a la entrada/salida (E/S).

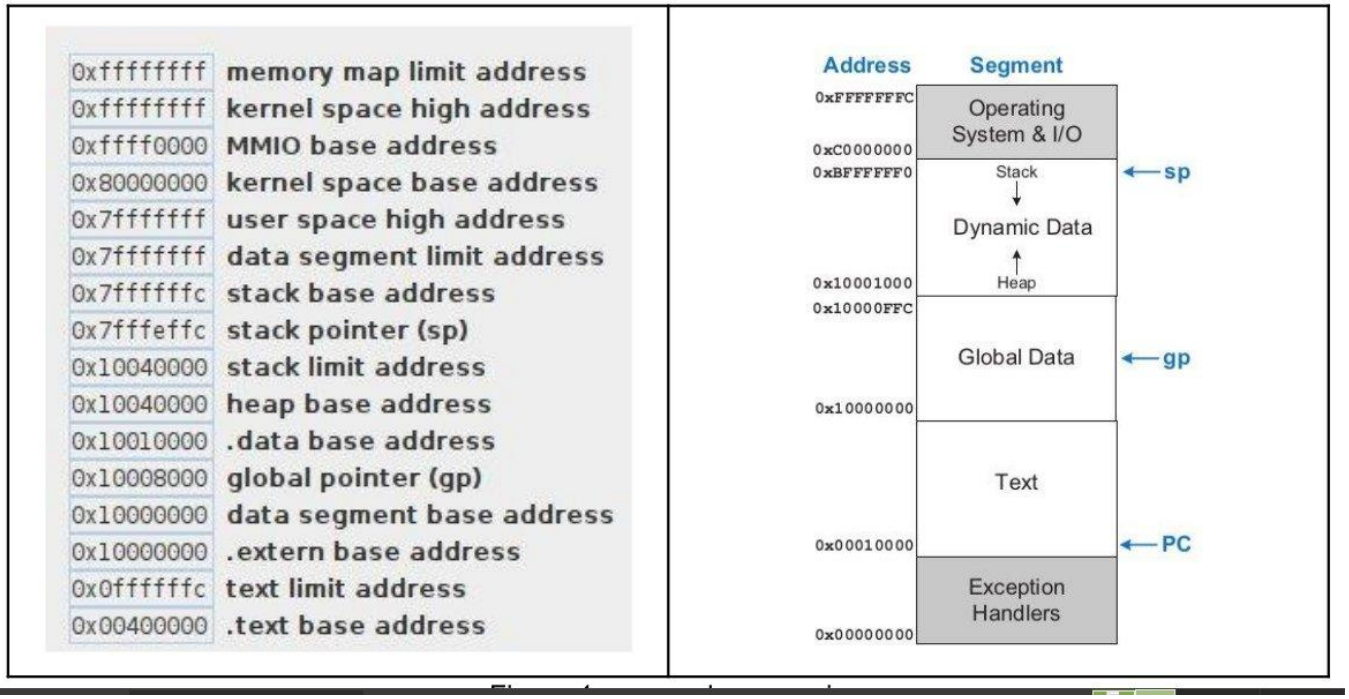

Figura 1 - mapa de memoria

Las siguientes secciones describen cada segmento. Presentamos dos ejemplos de mapa de memoria RISC-V aquí; esto es porque, RISC-V no define un mapa de memoria específico. Si bien el controlador de excepciones generalmente se encuentra en direcciones bajas o altas, el usuario puede definir dónde se colocan el .text (código y datos constantes), la E/S asignada a la memoria, la pila y los datos globales. Esto permite flexibilidad, especialmente con sistemas más pequeños, como dispositivos portátiles, donde solo se usa una parte del rango de memoria -*memoria física*- .

El segmento de texto

El segmento de texto almacena el programa de usuario en lenguaje de máquina. Además del código, puede incluir literales (constantes) y datos de solo lectura.

El segmento de datos globales

El segmento de datos globales almacena variables globales a las que, a diferencia de las variables locales, todas las funciones de un programa pueden acceder. Las variables locales se definen dentro de una función y solo esa función puede acceder a ellas; normalmente se encuentran en registros o en la pila.

Las variables globales se asignan en la memoria antes de que el programa comience a ejecutarse y, por lo general, se accede a ellas mediante el registro de puntero global gp (registro x3) que apunta a la mitad del segmento de datos globales. En este caso, gp es 0x10000800. Usando el desplazamiento con signo de 12 bits, los programadores pueden usar gp para acceder a todo el segmento de datos globales.

El segmento de datos dinámicos

El segmento de datos dinámicos contiene la pila y el Heap (montón). Los datos de este segmento no se conocen al inicio, pero se asignan y des-asignan dinámicamente a lo largo de la ejecución del programa. Al iniciarse, el sistema operativo configura el puntero de la pila (sp, registro x2) para apuntar a la parte superior de la pila, en este caso 0xBFFFFFF0.
La pila generalmente crece hacia abajo, como se muestra aquí. Las funciones usan la pila para guardar y restaurar registros. Se accede a cada marco de pila en orden de último en entrar, primero en salir LIFO.
El Heap almacena datos asignados por el programa durante el tiempo de ejecución. En C, las asignaciones de memoria se realizan mediante la función malloc; en C++ y Java, se usa new para asignar memoria. Al igual que un montón de ropa en el piso de un dormitorio, los datos del montón se pueden usar y desechar en cualquier orden.
El Heap normalmente crece hacia arriba desde la parte inferior del segmento de datos dinámicos.
Si la pila y el montón alguna vez se juntan, los datos del programa pueden corromperse. El asignador de memoria (memory allocator) garantiza que esto nunca suceda al devolver un error de falta de memoria si no hay espacio suficiente para asignar más datos dinámicos.

El controlador de excepciones, el sistema operativo y los segmentos de E/S
La parte más baja del mapa de memoria RISC-V de ejemplo está reservada para los controladores de excepciones (como veremos en la sección 2) y el código de inicio que se ejecuta al inicio. La parte más alta del mapa de memoria está reservada para el sistema operativo y las E/S asignadas a la memoria (como veremos en la sección 3).

2- Excepciones
Una excepción es como una llamada de función no programada provocada por un evento en el hardware o el software. Por ejemplo, el procesador puede recibir una notificación de que el usuario presiona una tecla en un teclado. El procesador puede detener lo que está haciendo, determinar que tecla se presionó, guardarla para referencia futura y luego reanudar el programa que se estaba ejecutando. Una excepción de hardware de este tipo desencadenada por un dispositivo de entrada/salida (E/S) como un teclado a menudo se denomina interrupción.
Alternativamente, el programa puede encontrar una condición de error causada por el software, como una instrucción indefinida. Las excepciones de software a veces se denominan trampas. Otras causas de excepciones incluyen reinicio e intentos de leer memoria inexistente. Como cualquier otra llamada de función, una excepción debe guardar la dirección de retorno, saltar a alguna dirección, hacer su trabajo, limpiarse y volver al programa donde lo dejó.

Modos de ejecución y niveles de privilegio
Un procesador RISC-V puede operar en uno de varios modos de ejecución con diferentes niveles de privilegio. Los niveles de privilegio dictan qué instrucciones se pueden ejecutar y a qué memoria se puede acceder. Los tres niveles principales de privilegios de RISC-V son el modo de usuario, el modo de supervisor y el modo de máquina, en orden creciente de privilegios.
El modo de máquina (modo M) es el nivel de privilegio más alto; un programa que se ejecuta en este modo puede acceder a todos los registros y ubicaciones de memoria. El modo M es el único modo de privilegio requerido y el único modo utilizado en procesadores sin sistema operativo (SO), incluidos muchos sistemas embebidos.
Las aplicaciones de usuario que se ejecutan sobre un sistema operativo generalmente se ejecutan en modo de usuario (modo U) y el sistema operativo se ejecuta en modo de supervisor (modo S); los programas de usuario no tienen acceso a registros privilegiados o ubicaciones de memoria reservadas para el sistema operativo. Los diferentes modos evitan que el estado clave se corrompa.

Discutimos las excepciones cuando se ejecuta en modo M. Las excepciones que ocurren en otros niveles son similares pero usan registros asociados con ese modo.

> Existe un cuarto nivel de privilegio denominado modo hipervisor (modo H) que admite la virtualización de máquinas, es decir, la aparición de varias máquinas (potencialmente con varios sistemas operativos) que se ejecutan en una sola máquina física. El modo H tiene más privilegios que el modo S, pero no tanto como el modo M

Controladores de excepciones

El controlador de excepciones utiliza cuatro registros de propósito especial, llamados registros de control y estado (CSR), para manejar (atender) una excepción: *mtvec, mcause, mepc y mscratch*.

El registro de dirección base del Tramp-Vector (vector trampa) de la máquina, *mtvec*, contiene la dirección del código para el controlador de excepciones.

Cuando ocurre una excepción, el procesador registra la causa de la excepción en *mcause*, almacena el PC de la instrucción de excepción en *mepc*, y salta al controlador de excepciones en la dirección preconfigurada en *mtvec*.

Después de saltar a la dirección en *mtvec* , el controlador de excepciones lee el registro *mcause* para examinar qué causó la excepción y responde adecuadamente (por ejemplo, leyendo el teclado en una interrupción de hardware).

Luego aborta el programa o regresa al programa ejecutando *mret*, instrucción de retorno de excepción de la máquina, que salta a la dirección en *mepc*. Mantener el PC de la instrucción de excepción en *mepc* es similar a usar ***ra*** para almacenar la dirección de retorno durante una instrucción ***jal***. Los controladores de excepciones deben usar registros de programa (*x1−x31*) para manejar excepciones, por lo que usan la memoria a la que apunta *mscratch* para almacenar y restaurar estos registros.

> RISC-V define una gran cantidad de CSR, todos los cuales deben inicializarse al inicio.

Instrucciones relacionadas con excepciones

Los controladores de excepciones usan instrucciones especiales para tratar las excepciones. Estas instrucciones se denominan instrucciones privilegiadas porque acceden a CSR. Son parte del conjunto de instrucciones base RV32I, Tabla 1. Los registros *mepc* y *mcause* no forman parte de los registros de programa RISC-V (*x1−x31*), por lo que el controlador de excepciones debe mover estos registros de propósito especial (CSR) a los registros del programa para leerlos y operar con ellos. RISC-V utiliza tres instrucciones para leer, escribir o leer y escribir CSR: **csrr** (leer CSR), **csrw** (escribir CSR) y **csrrw** (leer/escribir CSR).

Por ejemplo:

csrr t1, mcause lee el valor de *mcause* en t1;

csrw mepc, t2 escribe el valor en t2 en *mepc*;

csrrw t1, mscratch, t0 lee simultáneamente el valor de *mscratch* en t1 y escribe el valor de t0 en *mscratch*.

> El valor de *mcause* se puede clasificar como una interrupción o una excepción, como lo indica la columna más a la izquierda en la Tabla 1, que es el bit 31 de *mcause*. Los bits [30:0] de *mcause* contienen el código de excepción, que indica la causa de la interrupción o excepción.

Las excepciones pueden usar uno de los dos modos de manejo de excepciones: Directo o vectorizado. RISC-V generalmente usa el modo directo descrito aquí, donde todas las excepciones se bifurcan a la misma dirección, es decir, la dirección base codificada en bits 31:2 de *mtvec*.

En el modo vectorizado, las excepciones se bifurcan a un desplazamiento desde la dirección base, según la causa de la excepción. Cada desplazamiento está separado por una pequeña cantidad de direcciones, por ejemplo, 32 bytes, por lo que es posible que el código del controlador de excepciones deba saltar a un controlador de excepciones más grande para tratar la excepción.
El modo de excepción está codificado en bits 1:0 de *mtvec*; 00_2 es para modo directo y 01_2 para vectorizado.

Interrupción Bit 31 de *mcause*	**Código de Excepción** Bit 30:0	**Descripción**
1	3	Machine software interrupt
1	7	Machine timer interrupt
1	11	Machine external interrupt
0	0	Instruction address misaligned
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store address misaligned
0	7	Store access fault
0	8	Environment call from U-Mode
0	9	Environment call from S-Mode
0	11	Environment call from M-Mode

Tabla 1 - Códigos de excepción

Resumen de manejo de excepciones
En resumen, cuando un procesador detecta una excepción:
1. Salta a la dirección del controlador de excepciones que se encuentra en *mtvec*.
2. El controlador de excepciones guarda los registros en una pequeña pila a la que apunta *mscratch* y luego usa **csrr** (leer CSR) para buscar la causa de la excepción (codificada en *mcause*) respondiendo en consecuencia.
3. Cuando el controlador finaliza, opcionalmente incrementa mepc en 4, restaura los registros de la memoria y aborta el programa o regresa al código de usuario usando la instrucción **mret**, que salta a la dirección contenida en *mepc*.

Ejemplo 1 código del manejador de excepciones
Escriba un controlador de excepciones para manejar las siguientes dos excepciones: instrucción ilegal (mcause = 2) y dirección de carga desalineada (mcause = 4).
Si ocurre una instrucción ilegal, el programa simplemente debe continuar ejecutándose después de la instrucción ilegal. Ante una excepción de dirección de carga desalineada, el programa debería

cancelarse. Si ocurre cualquier otra excepción, el programa debe intentar volver a ejecutar la instrucción.

Solución

El controlador de excepciones comienza conservando los registros del programa que se sobrescribirán. Luego verifica cada causa de excepción y (1) continúa ejecutando justo después de la instrucción de excepción (es decir, en mepc + 4) en una excepción de instrucción ilegal, (2) aborta en una dirección de carga desalineada o (3) intenta volver a ejecutar. la instrucción de excepción (es decir, regresa a mepc) ante cualquier otra excepción.

Antes de volver al programa, el controlador restaura los registros que se sobrescribieron. Para abortar el programa, el controlador salta al código de salida ubicado en la etiqueta de salida (no se muestra). Para los programas que se ejecutan sobre un sistema operativo, la instrucción de salida **j,** puede reemplazarse por una llamada de entorno (ecall) con el código de retorno almacenado en un registro de programa como a0.

```
# Guardar los registros qué serán sobreescritos
        csrrw t0, mscratch, t0          # swap t0 and mscratch
        sw t1, 0(t0)                    # save t1 on mscratch stack
        sw t2, 4(t0)                    # save t2 on mscratch stack
# chequea la causa de la excepción
        csrr t1, mcause                 # t1 = mcause
        addi t2, x0, 2                  # t2 = 2 (código de excepción por instrucción ilegal)
illegalinstr:
        bne t1, t2, checkother          # branch if not an illegal instruction
        csrr t2, mepc                   # t2 = exception PC
        addi t2, t2, 4                  # increment exception PC by 4
        csrw mepc, t2                   # mepc = mepc + 4
        j done                          # restore registers and return
checkother:
        addi t2, x0, 4                  # t2 = 4 (código de excepción por carga desalineada)
        bne t1, t2, done                # branch if not a misaligned load
        j exit                          # exit program
# recupera los registros y retorna de la excepcion
done:
        lw t1, 0(t0)                    # restore t1 from mscratch stack
        lw t2, 4(t0)                    # restore t2 from mscratch stack
        csrrw t0, mscratch, t0          # swap t0 and mscratch
        mret                            # return to program (PC = mepc)
        …
        exit:
        ...
```

3- E/S mapeada en memoria

Los sistemas de entrada/salida (E/S) se utilizan para conectar una computadora con dispositivos externos llamados periféricos. En una computadora personal, los dispositivos suelen incluir teclados, monitores, impresoras y redes inalámbricas. En los sistemas embebidos, los dispositivos podrían incluir el elemento calefactor de una tostadora, el sintetizador de voz de una muñeca, el inyector de combustible de un motor, los motores de posicionamiento del panel solar de un satélite, etc.

Un procesador accede a un dispositivo de E/S utilizando los buses de dirección y datos de la misma manera que accede a la memoria.
Este tema proporciona ejemplos concretos de dispositivos de E/S.
La sección 3-1 muestra los principios básicos de la interfaz de un dispositivo de E/S con un procesador y cómo acceder a él desde un programa.

3-1 Interfaz E/S mapeada en memoria
Recuerde de la Sección 1 que una parte del espacio de direcciones está dedicada a los dispositivos de E/S en lugar de a la memoria. Por ejemplo, suponga que las direcciones logicas en el rango de 0x20000000 a 0x20FFFFFF se usan para E/S. A cada dispositivo de E/S se le asigna una o más direcciones de memoria en este rango. Un store (almacenamiento) a la dirección especificada envía datos al dispositivo. Un load (carga) recibe datos del dispositivo. Este método de comunicación con dispositivos de E/S se denomina E/S mapeada en memoria.

En un sistema con E/S mapeadas en memoria, una carga o almacenamiento puede acceder a la memoria o a un dispositivo de E/S. La Figura 2 muestra el hardware necesario para admitir dos dispositivos de E/S mapeados en memoria. Un decodificador de direcciones determina qué dispositivo se comunica con el procesador. Utiliza las señales Address y MemWrite para generar señales de control para el resto del hardware. El multiplexor ReadData selecciona entre la memoria y los diversos dispositivos de E/S. Los registros habilitados para escritura contienen los valores escritos en los dispositivos de E/S.

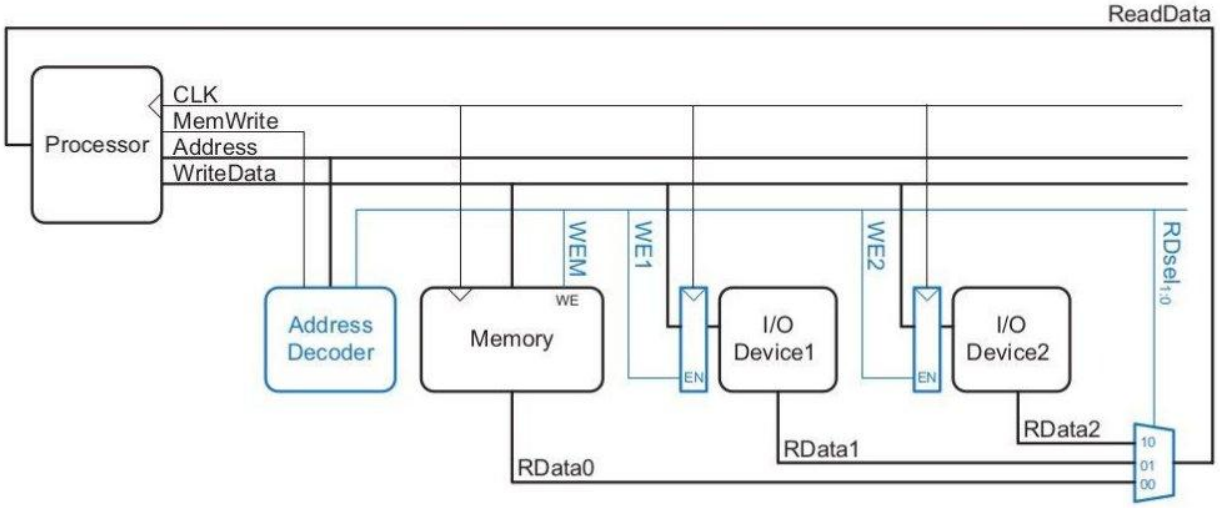

Figura 2 - Hardware para dispositivos mapeados en memoria.

Ejemplo 2
Comunicación con dispositivos de E/S
Suponga que al dispositivo de E/S 1 de la figura 2 se le asigna la dirección de memoria 0x20001000 y 0x20001001.
Muestre el código ensamblador RISC-V para escribir el valor 7 en el dispositivo de E/S 1 y para leer el valor de salida del dispositivo de E/S 1.

Solución
El siguiente código ensamblador RISC-V escribe el valor 7 en el dispositivo de E/S 1 (puerto de entrada). Entonces, la instrucción li s1, ioadr se convierte en li s1, 0x20001000.

iadr: 0x20001000
oadr: 0x20001001
…

```
li s0, 7
li s1, iadr
sw s0, 0(s1)
```

El decodificador de direcciones detecta la dirección 0x20001000 y MemWrite = 1, por lo que activa WE1, habilitación de escritura para el registro del Dispositivo 1. En el siguiente flanco del reloj, el valor del bus WriteData, 7, se escribe en el registro, cuya salida se conecta a los pines de entrada del dispositivo de E/S 1.
Para leer desde el dispositivo de E/S 1, el procesador ejecuta el siguiente código ensamblador RISC-V.

```
li s1, oadr
lw s0, 0(s1)
```

El decodificador de direcciones detecta la dirección 0x20001001, por lo que establece RDsel 1:0 a 01.
Por lo tanto, el multiplexor selecciona RData1, los datos leídos del Dispositivo 1, y los conecta al bus ReadData, cuyo valor se carga luego en s0 en el procesador.
Las direcciones asociadas con los dispositivos de E/S a menudo se denominan registros de E/S porque pueden corresponder con registros físicos en el dispositivo de E/S como los que se muestran en la Figura 2.
