Proyecto de Ingeniería de Computadores (PEC)

Subetapa 7.3: Modo Sistema

En este capítulo añadiremos el soporte para que el sistema operativo pueda ejecutar tareas de forma privilegiada. Es decir, el procesador deberá tener unas características que haga que determinadas instrucciones sólo se puedan ejecutar en el modo privilegiado del procesador y así permitir definir de forma segura rutinas de atención a las interrupciones, llamadas a sistema y otras características que un procesador moderno debe tener.

Añadiremos un modo privilegiado (modo sistema). Este modo tiene acceso a toda la memoria y hardware, así como la posibilidad de ejecutar todo el repertorio de instrucciones del lenguaje, mientras que el modo no privilegiado (modo usuario) sólo puede utilizar un subconjunto de instrucciones. La razón de esta separación es la protección: como una aplicación de usuario no puede realizar acciones que puedan alterar el estado de la máquina, se garantiza la seguridad del sistema y de las demás aplicaciones.

Los principales cambios de esta etapa serán:

- Distinción entre modo usuario y sistema (o privilegiado).
- Nuevas excepciones relacionadas con el modo de ejecución.
- Una nueva instrucción de llamada al sistema operativo (**`CALLS`**).
- Algunos cambios en los registros de sistema

Llamadas al sistema

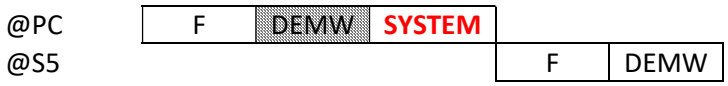
En esta etapa se tratará de desarrollar el apoyo para que los programas de usuario puedan hacer llamadas al sistema. Las llamadas al sistema son vitales para que los programas puedan realizar tareas privilegiadas con el control del sistema operativo, sin cambiar de modo de ejecución.

Para hacer esto, el SISA dispone de una instrucción llamada **`CALLS`**, que cambia de modo de ejecución en el procesador. Pasa de modo usuario a modo sistema y salta a la rutina de servicio general del sistema operativo, almacenada en el registro S5. Esta instrucción hace más cosas. En orden, realiza el siguiente proceso:

1. Se guarda el registro S0 la palabra de estado, que se encuentra en el S7. Si el procesador se encontraba en modo sistema, el modo no cambia, pero si estaba en modo usuario, se salva el registro y se pasa a modo sistema.
2. Se guarda el registro S1 la dirección del PC actualizado para hacer el retorno de la rutina de servicio del sistema operativo.
3. Se escribe en el registro S2 el código del tipo de evento que ha ocurrido. En el caso de una llamada al sistema se trata del número 14.
4. **Copia el contenido del registro fuente (Ra) de la instrucción `CALLS`, donde hay almacenado el número de servicio del sistema operativo que se desea ejecutar, en el registro S3.**
5. Se modifica la palabra de estado en el registro S7, pasando a modo sistema e inhibiendo las interrupciones (bit 0 a 1 y bit 1 a 0).
6. Se escribe en el PC el contenido del registro S5, que tiene la dirección del único punto de entrada al sistema operativo.
7. Finalmente se pasa a ejecutar la instrucción apuntada por el PC.

Esta secuencia de pasos es bastante parecida a las operaciones que se llevan a cabo en el caso de una interrupción o de una excepción, por lo tanto no debemos despreciar esta situación. En este caso, para realizar todas estas operaciones usaremos el nuevo estado del procesador (denominado SYSTEM) para que almacene todos estos datos en los registros de sistema y se ejecute después del ciclo típico de ejecución, en el caso de que en este ciclo se produzca llamada a sistema.

Durante la ejecución de la instrucción **CALLS** no hace falta realizar ninguna tarea en el estado de DEMW pero igualmente podemos ejecutar ese estado para simplificar la implementación de la unidad de control.



Para salir de la llamada al sistema ejecutamos siempre la instrucción **RETI**. Esta instrucción carga la palabra de estado en el registro S7 con la que está salvada en el registro S0. También carga el PC del modo usuario con el valor salvado el registro S1.

Modos de ejecución

La arquitectura tiene dos modos de ejecución, modo usuario y modo sistema (o privilegiado). Algunas instrucciones sólo se pueden ejecutar cuando el procesador se encuentra en modo sistema, provocando una excepción en caso de pretender ejecutarlas en modo usuario. La memoria está dividida en dos partes: usuario y sistema. Si en modo usuario se pretende acceder a la parte de memoria de sistema, se produce una excepción.

Memoria

Esta arquitectura tiene un único espacio de direccionamiento lógico (o virtual) de memoria, que se usa tanto para código como para datos. Una dirección lógica de memoria consta de 16 bits y la unidad de direccionamiento es el byte: se pueden direccionar hasta 2^16 bytes de memoria. Las direcciones lógicas, que son las que se ven en el programa en lenguaje máquina, son trasladadas por el controlador de memoria a direcciones físicas de la memoria real. En lo que sigue, cuando se hable del espacio de direccionamiento de memoria o de una dirección de memoria, mientras no se indique lo contrario nos referimos al espacio o a la dirección lógica.

El espacio de direccionamiento de memoria está divido en dos partes del mismo tamaño, la mitad baja de las direcciones (bit 15 de la dirección con valor 0) son accesibles en modo usuario y modo sistema, mientras que la mitad alta de las direcciones (bit 15 de la dirección con valor 1) sólo son accesibles en modo sistema:

- Memoria Usuario: de `0x0000` a `0x7FFF`
- Memoria Sistema: de `0x8000` a `0xFFFF`

Cualquier acceso, a instrucciones o a datos, que se realice en modo usuario a la zona de memoria de sistema produce una excepción del tipo "acceso a memoria protegida".

Listado de las nuevas excepciones

Al añadir el modo sistema se pueden generar nuevas excepciones. Las nuevas excepciones que deberemos tratar serán las siguientes:

- Excepción 11: Acceso a memoria protegida. El programa de usuario está ejecutándose en modo no privilegiado e intenta leer (load) o escribir (store) en una página que pertenece al sistema operativo.
- Excepción 13: Excepción de protección. Se lanza cuando se está intentado ejecutar una instrucción protegida y el procesador está en modo usuario.
- Excepción 14: Llamada al sistema. Esta no es propiamente una excepción, sino que se produce cuando se ejecuta la instrucción **CALLS**.

Cambios en el uso de los registros de sistema

Los registros de sistema no se pueden leer ni escribir mediante ninguna instrucción cuando el procesador está en modo usuario, es decir, cuando PSW.M = 0. Se acceden explícitamente mediante las instrucciones privilegiadas **RDS** y **WRS**, que sólo se pueden ejecutar en modo sistema, cuando PSW.M = 1. No obstante, como resultado de la ejecución de una instrucción en modo usuario se pueden modificar estos registros, por ejemplo si se produce una excepción.

Los registros de sistema que hay que modificar su uso añadiendo algo son los siguientes:

S0. Contiene la palabra de estado que tenía el sistema cuando se produjo una excepción (interna), llamada a sistema (**CALLS**) o interrupción (externa). Cuando se produce uno de estos eventos, se salva en S0 la palabra de estado actual, que se encuentra en S7, ya que acto seguido se cambia la palabra de estado actual, para pasar a modo sistema e inhibir las interrupciones (externas).

S1. Contiene la dirección de retorno después de una interrupción (externa), una excepción (interna) o la ejecución de la instrucción de llamada al sistema operativo (**CALLS**). Cuando se produce uno de estos eventos, se salva en S1 el PC actualizado, ya que acto seguido se copia en el PC el registro S5, que contiene la dirección del código de entrada al sistema operativo.

S2. Contiene, en sus cuatro bits de menor peso, un código binario que puede tomar valores de 0 a 15 que identifica el tipo de evento que se ha producido: tipo concreto de excepción (interna), llamada a sistema (**CALLS**) o interrupción (externa). El resto de bits de S1 están siempre a 0. Por ahora los códigos usados son:

- 0: Excepción de tipo "Instrucción ilegal".
- 1: Excepción de tipo "Acceso a memoria mal alineado".
- 2: Excepción de tipo "Overflow en operación de coma flotante".
- 11: Excepción de tipo "Acceso a memoria protegida".
- 13: Instrucción protegida.
- 14: **CALLS**
- 15: Interrupción (externa)

S3. **Contiene una copia del registro que contiene el número de servicio de sistema operativo que se quiere ejecutar con la instrucción CALLS.**

S5. Contiene la dirección de memoria de sistema donde se encuentra la siguiente instrucción a ejecutar después de producirse una excepción (interna), llamada a sistema (**CALLS**) o interrupción (externa). En esta dirección de memoria comienza el código de la rutina de servicio genérica (RSG) encargado de identificar la causa de excepción o interrupción y en cada caso llamar a la rutina de servicio de excepción (RSE) o a la rutina de servicio de interrupción (RSI) específica correspondiente.

S7. Es el PSW (Processor Status Word). Contiene el estado del procesador en todo momento. Cada bit tiene el siguiente significado:

Bit M (PSW<0>): Bit que indica el modo de trabajo del procesador. Si el bit es 0, entonces el procesador se encuentra en modo usuario y la ejecución de ciertas instrucciones privilegiadas y/o el acceso a la memoria del sistema están restringidos, pudiendo llegar, según el caso, a provocar una excepción. Si el bit es 1, el procesador se encuentra en modo sistema, o privilegiado, pudiendo ejecutar cualquier instrucción y acceder a cualquier parte de la memoria.

Bit I (PSW<1>): Bit que indica si las interrupciones están permitidas (en cuyo caso vale 1) o si están inhibidas (en cuyo caso vale 0). Este bit se modifica con las instrucciones privilegiadas **EI** (Enable Interrupt) y **DI** (Disable Interrupt), que lo ponen a 1 o a 0 respectivamente y la instrucción **RETI** que restaura la palabra de estado y retorna de una excepción, interrupción o de una llamada a sistema, **CALLS**.

También, se pone a 0 cuando se produce una excepción, interrupción o se ejecuta la instrucción de llamada a sistema, **CALLS**.

Bit V (PSW<2>): Bit que indica si la excepción de overflow en operación de coma flotante está permitida (en cuyo caso vale 1) o si está inhibida (en cuyo caso vale 0).

Modificaciones en los componentes de procesador

Primero analizaremos los cambios que hay que hacer en los módulos de ejecución del procesador como hemos hecho habitualmente hasta ahora.

Módulo de control

Las instrucciones que solo se permiten ejecutar en modo privilegiado son **RDS**, **WRS**, **EI**, **DI**, **RETI** y **GETIID**. Habrá que actualizar la unidad de control para que detecte si se están intentando ejecutar en modo usuario y en este caso debería generar una excepción.

Hay otras instrucciones que también deberían ser privilegiadas como son la gestión de los dispositivos de entrada (**IN**/**OUT**) y la de parar el procesador **HALT** (no es normal que un usuario pueda detener completamente el procesador). Normalmente estas acciones se hacen a través de servicios del sistema operativo, pero como aún no lo hemos implementado las dejaremos que se puedan ejecutar en modo usuario y así se nos facilitará enormemente realizar los juegos de prueba.

La unidad de control debe generar unas cuantas señales nuevas o adaptar, en caso de que sea necesario, las ya existentes para implementar la nueva instrucción **CALLS**. La siguiente tabla muestra la codificación de la instrucción **CALLS**.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
1 0 1 0	0 0 0	Ra	0 0 0 1 1 1	**CALL** System	CALLS	

La instrucción **CALLS** se utiliza para llamadas al sistema operativo. Salva la dirección de retorno en S1 y se prepara para pasar a ejecutar la rutina del sistema operativo que se encuentra en la dirección contenida en el registro de sistema S5. El procesador, al finalizar la ejecución de esta instrucción, inhibe las interrupciones, poniendo S7<1> = 0, y pasa a modo sistema, poniendo S7<0> = 1. En el contenido del registro Ra se encuentra codificado el servicio a realizar. Los servicios posibles y qué registros se usan para codificarlos depende de la implementación del sistema operativo, lo que no forma parte de la definición de la arquitectura. **El procesador debe copiar el contenido del registro Ra en el registro S3 de sistema. El sistema operativo deberá hacer un "switch" sobre el valor contenido en S3 para saber qué servicio ejecutar**. Produce una excepción de tipo "instrucción ilegal" si se ejecuta en modo sistema. Esta instrucción solo debe poderse ejecutar en modo usuario.

Banco de registros

También tenemos que cambiar ligeramente el banco de registros. Cuando se produzca una excepción de tipo 14 (**CALLS**) el banco de registros debe almacenar en el registro S2 el código correspondiente al tipo de excepción producido. Si habéis implementado un *módulo de excepciones*, este será el encargado de gestionar la nueva excepción. **También deberá ser capaz de copiar el contenido del un registro de usuario (que puede ser cualquier registro ya que el registro fuente está indicado en el formato de la instrucción CALLS) al registro S3. Si os es más práctico, podríais hacer esta acción durante el ciclo DEMW de la instrucción CALLS.**

Ahora, ya no inician los registros en cualquier estado sino que hay que preocuparse de arrancar el procesador en el modo de sistema. Por lo tanto, la palabra de estado (almacenada en el registro S7) tenemos que cambiarla para iniciar con el bit de menor peso a 1 (modo privilegiado). Si no el sistema operativo no podría almacenar la dirección de la rutina de servicio de interrupción en S5 ya que para ello se debe ejecutar una instrucción protegida.

¿Qué pasa con la memoria de la VGA que ocupa un espacio de direccionamiento de sistema?

De momento haremos que no genere ninguna excepción y así el usuario pueda leer y escribir en la memoria de la pantalla, por los mismos motivos por los que hemos dejado las instrucciones **`IN`**, **`OUT`** y **`HALT`** en modo usuario.

Juegos de pruebas

Realizad un juego de pruebas que provoque y trate todas estas excepciones.

Anexo A: Esqueleto del código de la RSG

Un fragmento/esqueleto del código con la RSG completa en SISA, con interrupciones, excepciones y llamadas a sistema, quedaría más o menos de la siguiente forma.

Este fragmento asume que el número de servicios que tendrá el S.O. será de únicamente 8 y por tanto sólo usa los 3 bits de menor peso del registro que se use en la llamada CALLS para indexarlos. Además falta por escribir las tareas de los servicios, excepciones e interrupciones concretas que queramos tratar.

```
; Incluir las macros necesarias
.include "macros.s"

.set PILA, 0x4000      ; una posición de memoria de una zona no ocupada para usarse como PILA

; seccion de datos
.data
      .balign 2                     ; por si acaso, pero no debería ser necesario
      interrupts_vector:
            .word RSI_default_resume ; 0 Interval Timer
            .word RSI_default_resume ; 1 Pulsadores (KEY)
            .word RSI_default_resume ; 2 Interruptores (SWITCH)
            .word RSI_default_resume ; 3 Teclado PS/2

      exceptions_vector:
            .word RSE_default_halt   ; 0 Instrucción ilegal
            .word RSE_default_halt   ; 1 Acceso a memoria no alineado
            .word RSE_default_resume ; 2 Overflow en coma flotante
            .word RSE_default_resume ; 3 División por cero en coma flotante
            .word RSE_default_halt   ; 4 División por cero
            .word RSE_default_halt   ; 5 No definida
            .word RSE_excepcion_TLB  ; 6 Miss en TLB de instrucciones
            .word RSE_excepcion_TLB  ; 7 Miss en TLB de datos
            .word RSE_excepcion_TLB  ; 8 Página inválida al TLB de instrucciones
            .word RSE_excepcion_TLB  ; 9 Página inválida al TLB de datos
            .word RSE_default_halt   ; 10 Página protegida al TLB de instrucciones
            .word RSE_default_halt   ; 11 Página protegida al TLB de datos
            .word RSE_default_halt   ; 12 Página de sólo lectura
            .word RSE_default_halt   ; 13 Excepción de protección

      call_sys_vector:
            .word RSE_default_resume ; 0 Hay que definirla en el S.O.
            .word RSE_default_resume ; 1 Hay que definirla en el S.O.
            .word RSE_default_resume ; 2 Hay que definirla en el S.O.
            .word RSE_default_resume ; 3 Hay que definirla en el S.O.
            .word RSE_default_resume ; 4 Hay que definirla en el S.O.
            .word RSE_default_resume ; 5 Hay que definirla en el S.O.
            .word RSE_default_resume ; 6 Hay que definirla en el S.O.
            .word RSE_default_resume ; 7 Hay que definirla en el S.O.


; sección de código
.text
     ;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
     ; Inicialización
     ;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
     $MOVEI r1, RSG
     wrs    s5, r1       ; inicializamos en S5 la dirección de la rutina de atención a la interrupción
     $MOVEI r7, PILA     ; inicializamos R7 como puntero a la pila
     $MOVEI r6, inici    ; dirección de la rutina principal
     jmp    r6

     ;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
     ; Rutinas de servicio por defecto
     ;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
     RSE_default_halt:   HALT
     RSE_default_resume: JMP R6

     RSE_excepcion_TLB:    ; fragmento de código
          ; falta el código de la tarea a hacer
          rds   R2, S1      ; hay que volver a ejecutar la instrucción que ha fallado
```

```
        addi  R2, R2, -2
        wrs   S1, R2
        JMP R6

;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
; Rutina de servicio de interrupción
;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
RSG: ; Salvar el estado
            $push  R0, R1, R2, R3, R4, R5, R6
            rds    R1, S0
            rds    R2, S1
            rds    R3, S3
            $push  R1, R2, R3
            rds    R1, S2                   ;consultamos el contenido de S2
            movi   R2, 14
            cmpeq  R3, R1, R2               ;si es igual a 14 es una llamada a sistema
            bnz    R3, __call_sistema       ;saltamos a las llamadas a sistema si S2 es igual a 14
            movi   R2, 15
            cmpeq  R3, R1, R2               ;si es igual a 15 es una interrupción
            bnz    R3, __interrupcion       ;saltamos a las interrupciones si S2 es igual a 15
  __excepcion:
            movi   R2, lo(exceptions_vector)
            movhi  R2, hi(exceptions_vector)
            add    R1, R1, R1               ;R1 contiene el identificador de excepción
            add    R2, R2, R1
            ld     R2, 0(R2)
            jal    R6, R2
            bz     R3, __finRSG
  __call_sistema:
            rds    R1, S3                   ;S3 contiene el identificador de la llamada a sistema
            movi   R2,7
            and    R1, R1, R2     ;nos quedamos con los 3 bits de menor peso limitar el número de servicios definidos en el S.O.
            add    R1, R1, R1
            movi   R2, lo(call_sys_vector)
            movhi  R2, hi(call_sys_vector)
            add    R2, R2, R1
            ld     R2, 0(R2)
            jal    R6, R2
            bnz    R3, __finRSG
  __interrupcion:
            getiid R1
            add    R1, R1, R1
            movi   R2, lo(interrupts_vector)
            movhi  R2, hi(interrupts_vector)
            add    R2, R2, R1
            ld     R2, 0(R2)
            jal    R6, R2
  __finRSG: ;Restaurar el estado
            $pop  R3, R2, R1
            wrs   S3, R3
            wrs   S1, R2
            wrs   S0, R1
            $pop  R6, R5, R4, R3, R2, R1, R0
            reti


;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
; Rutina principal
;*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
inici: $MOVEI r6, inici             ;bucle infinito a la espera de que lleguen interrupciones
       jmp    r6
       halt
```