Lazy stacking

Mg. Ing. Gonzalo E. Sanchez
Esp. Ing. Hanes N. Sciarrone
MSE - 2023

Implementación de Sistemas Operativos (I)

Lazy Stacking

- Generalidades
- Funcionalidad lazy stacking
- Lazy stacking en OS

Generalidades

Generalidades

- El núcleo Cortex-M4 puede ser fabricado con la variante F.
- Es el caso del STM32F429ZIT6 utilizado en la NUCLEO-F429ZI.
- Esta variante tiene registros de coprocesador que van desde S0-S31 y algunos más mapeados en memoria.

Generalidades

- Floating-point Status and Control Register (FPSCR)
  - Es un registro especial de la FPU.
  - No está mapeado en memoria.
  - Contiene bits de estado de operaciones y control de la FPU.
  - Para acceder/modificar su contenido se utilizan dos instrucciones especiales (VMRS y VMSR).

Generalidades

- Coprocessor Access Control Register (CPACR)
  - Registro mapeado en memoria (0xE000ED88).
  - Los bits [23:20] deben ser seteados a 0xF para habilitar la FPU.
  - Por defecto, luego de un reset, la FPU no está habilitada.
- Floating-point Context Control Register (FPCCR)
  - Registro mapeado en memoria (0xE000EF34).
  - Controla el comportamiento del guardado de contexto.
  - Por defecto, habilita el mecanismo de *lazy stacking*.

Generalidades

- Floating-point Context Address Register (FPCAR)
  - Registro mapeado en memoria (0xE000EF38).
  - Contiene la dirección del espacio reservado que se asigna en el stack frame para los registros de punto flotante.
  - Dirección apunta a una sección dentro del stack utilizado al momento de la excepción (generada automáticamente por el hardware).
  - Registros aún no han sido almacenados en el stack frame al momento de entrar en la excepción.
  - Utilizado para el mecanismo de *lazy stacking*.

Generalidades

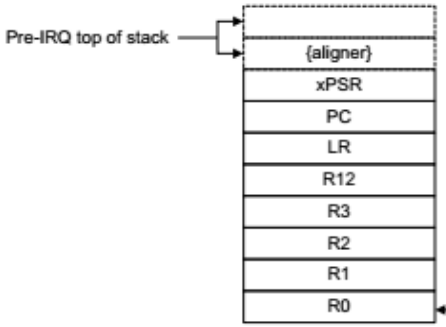
- Además de todos estos registros, el registro CONTROL implementa algunos bits para el control de la FPU.
- El bit CONTROL[2] se define como FPCA (Floating-Point Context Active).
- Este bit se setea automáticamente cuando se utiliza la FPU.
- Se limpia cuando se cambia de contexto (ej: ISR).

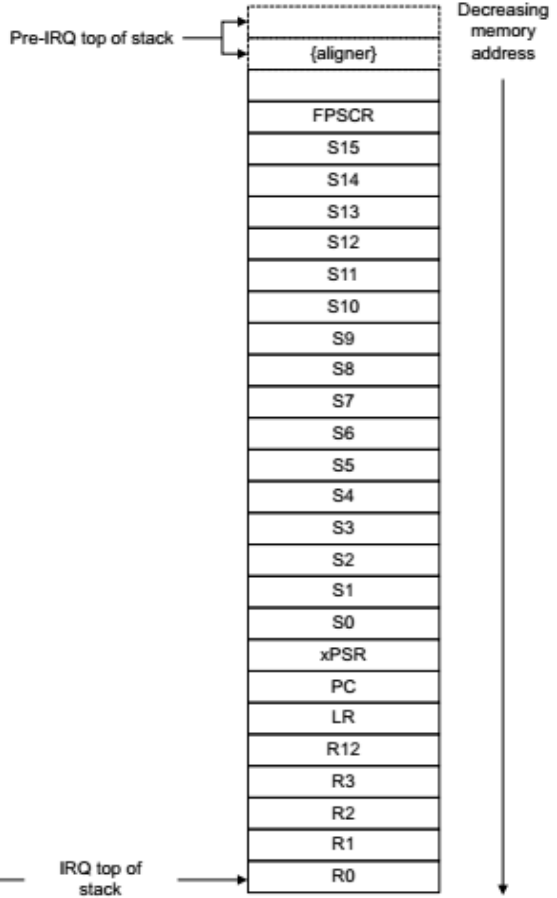
Generalidades

- El valor EXEC_RETURN tiene un bit adicional.
- Cuando el bit EXEC_RETURN[4] = 0, en el stack frame hay registros de la FPU.
- Cuando el bit EXEC_RETURN[4] = 1, es el caso opuesto al anterior.
- Si FPCA = 1, entonces el stack frame se compone del stack frame básico más los registros S0-S15 y FPSCR.

Generalidades

Stack Frame sin registros de FPU





Stack Frame con registros de FPU

Generalidades

- El stacking de los registros de la FPU tiene los siguientes efectos:
  - Aumento de tamaño del stack frame.
  - Potencial incremento del tiempo de latencia para atención de interrupciones.
  - Incremento del tiempo necesario para hacer cambio de contexto en un OS.
- Para sistemas baremetal, el stack frame automático junto con el stacking del compilador para las funciones, es suficiente.

Generalidades

- En el caso de un OS hay que prever el stacking de los registros que no están en el stack frame.
- Esto es de vital importancia para un cambio de contexto funcional.
- Atendiendo a los efectos del stacking de tantos registros, el OS debe determinar cuando es necesario hacerlo.

Generalidades

- Durante el cambio de contexto el OS debe:
  a. Determinar si la tarea está usando la FPU (EXEC_RETURN[4] = 0).
  b. Guardar el contexto del FPU (o no) en base al punto anterior.
  c. Restaurar el contexto de la FPU si es requerido por la tarea siguiente.
  d. Ir a la tarea siguiente utilizando el codigo EXEC_RETURN correspondiente (depende si la tarea usa FPU o no).

Funcionalidad Lazy Stacking

Lazy Stacking

- Cortex-M4F presenta una funcionalidad llamada Lazy Stacking.
- Consiste en evitar un incremento en el tiempo de latencia en interrupciones.
- Se logra evitando el stacking de los registros de FPU si no es necesario.
- Esto se da cuando el handler de IRQ no utiliza FPU.
- Si el software interrumpido no utiliza FPU.

Lazy Stacking

- El término *lazy* se utiliza porque de ser necesario, el stacking se hace al momento anterior a la primer utilización de la FPU.
- Ejemplo: Tanto el software interrumpido como el IRQ handler utilizan FPU.
- El stacking dentro del handler IRQ se hará justo antes de la primer instrucción que involucra la FPU.

Lazy Stacking

- Cuando ocurre una excepción y lazy stacking está activado, se reserva lugar extra en el stack para S0-S15 y FPSCR.
- El stacking de estos 17 registros queda pendiente. Bit Lazy State Preservation Active (LSPACT) = 1.
- EXC_RETURN[4] = 0 indicando que hay espacio reservado en el stack frame para los registros FPU.
- Esto desemboca en dos posibles casos.

Lazy Stacking

- Caso 1: El handler de la excepción no utiliza FPU:
  - LSPACT = 1 hasta el final del handler.
  - Al llegar al retorno se detecta LSPACT = 1 y EXC_RETURN[4] = 0.
  - Stack frame reservo espacio pero jamás se hizo push de los registros.
  - POP de registros de FPU es ignorado. No hay nada que recuperar.

Lazy Stacking

- Caso 2: El handler de la excepción utiliza FPU en algún punto:
  - CPU stall al detectar primer instrucción involucrando FPU.
  - PUSH de los registros S0-S15 y FPSCR.
  - LSPACT = 0.
  - Al finalizar el handler se detecta LSPACT = 0 y EXC_RETURN[4] = 0.
  - POP de registros de FPU correspondientes.

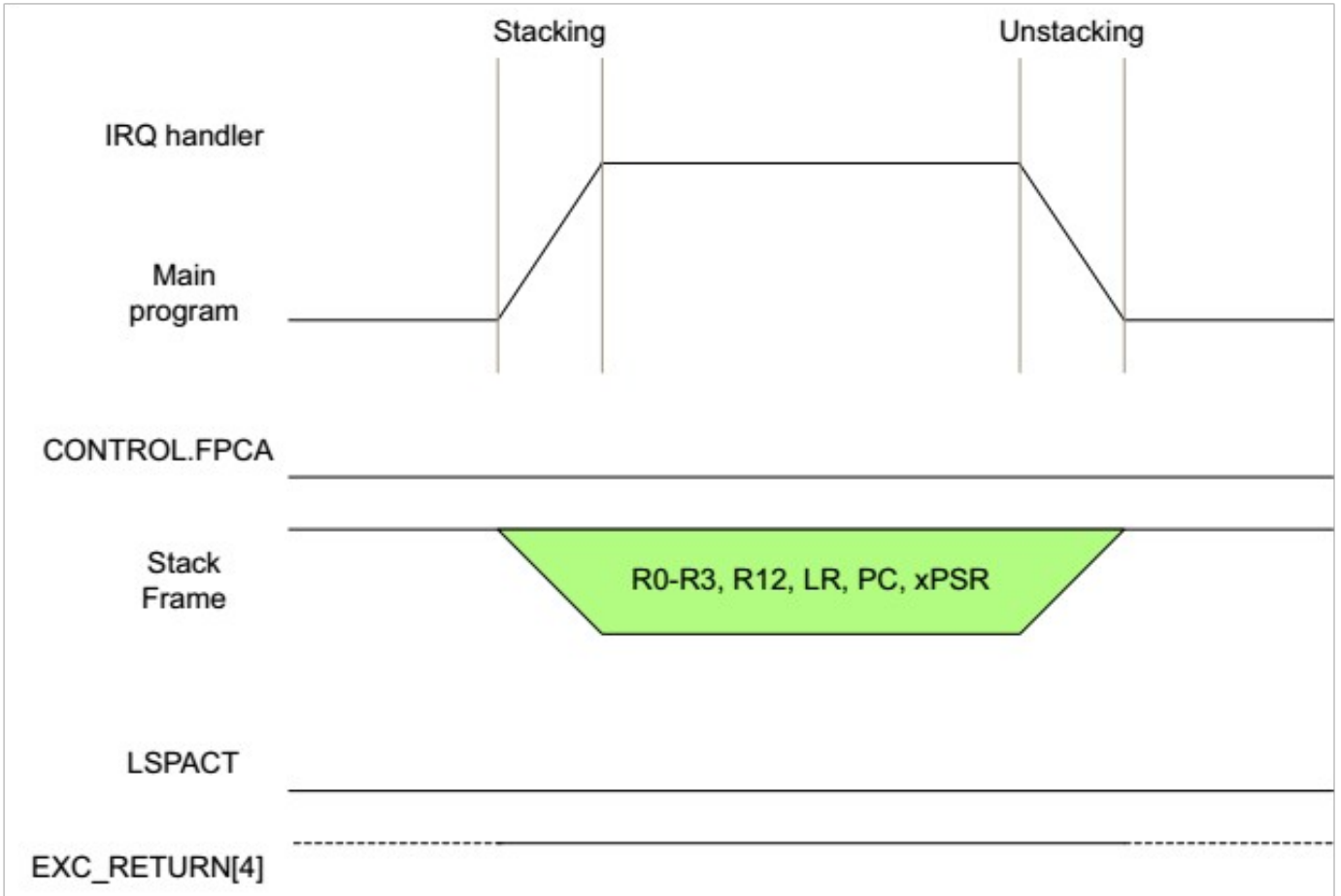
Lazy Stacking

- El lazy stacking puede desactivarse en cualquier momento de ejecución.
- Conlleva a que el stack frame consista siempre en los registros de core y registros de FPU.
- Latencia incrementada para atención de interrupciones.

Lazy Stacking

- A continuación varios ejemplos de casos de utilización.
- Los diagramas temporales muestran las operaciones de stacking y unstacking.
- **Ejemplo 1**: Ni el programa interrumpido, ni el handler de IRQ utilizan la FPU.
- Como FPCA = 0, no se reserva espacio para los registros de FPU.
- Stack frame igual al visto hasta el momento.

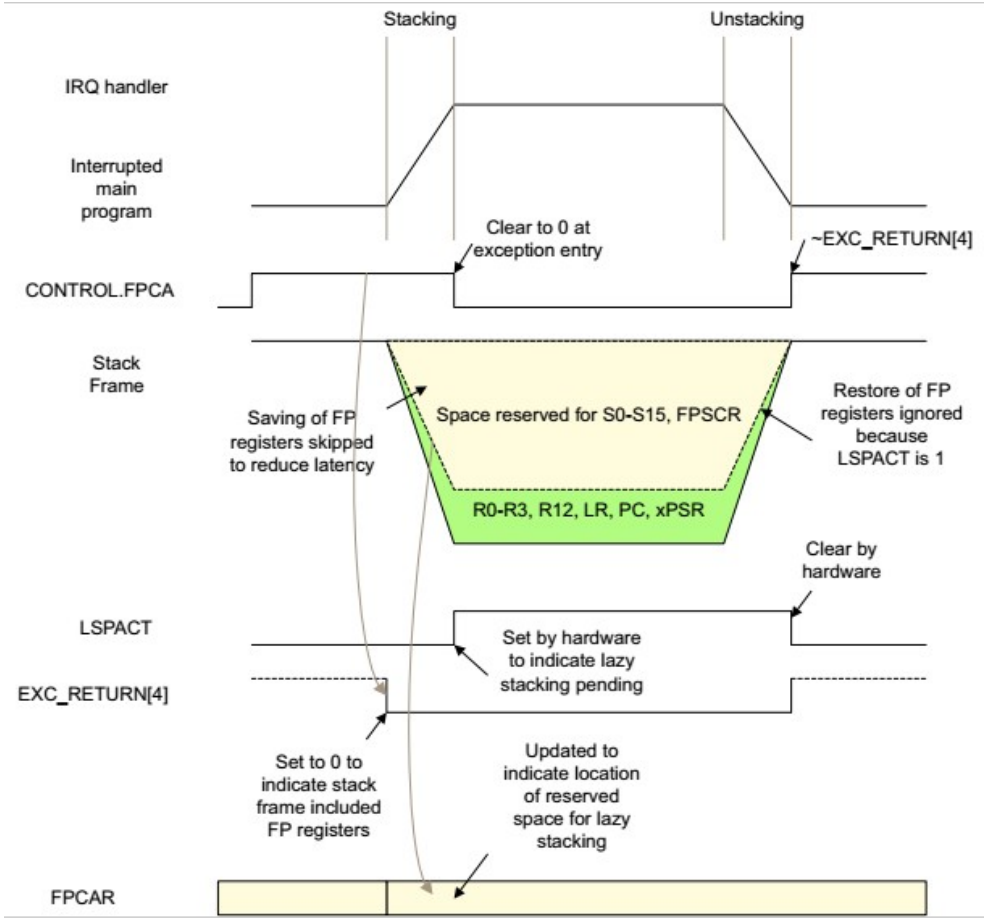
Lazy Stacking



Lazy Stacking

- **Ejemplo 2**: Se utilizó la FPU en algún punto entre el reset y la ocurrencia de IRQ. Handler no utiliza la FPU
- FPCA = 1 al momento de IRQ. Se reserva espacio para los registros de FPU.
- LSPACT = 1 hasta el final del handler. No se hace push de los registros de FPU.
- Stack frame igual al visto hasta el momento, pero desplazado por el espacio reservado.

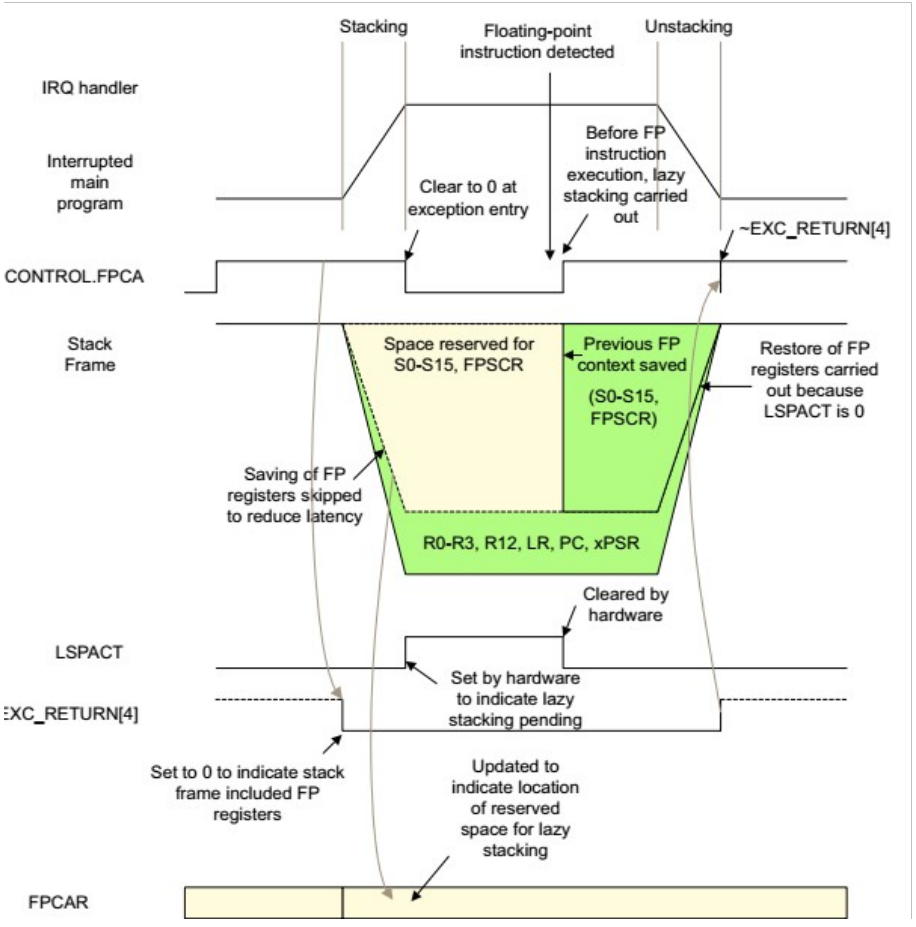
Lazy Stacking



Lazy Stacking

- **Ejemplo 3**: Se utilizó la FPU en algún punto entre el reset y la ocurrencia de IRQ. Handler utiliza la FPU
- FPCA = 1 al momento de IRQ. Se reserva espacio para los registros de FPU.
- LSPACT = 1 hasta la primer instrucción que utiliza la FPU.
- Stack frame contiene los registros de FPU.
- **NOTA**: Para cumplir AAPCS se debe hacer push manualmente de los registros S16-S31

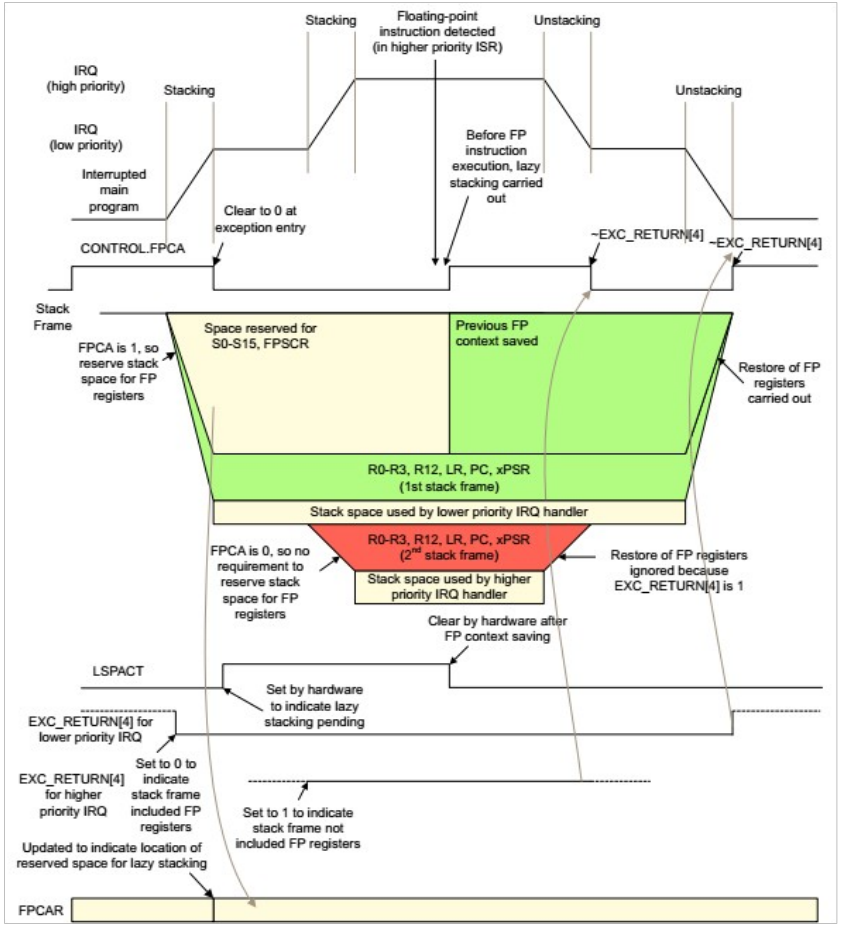
Lazy Stacking



Lazy Stacking

- **Ejemplo 4**: Se utiliza la FPU en software interrumpido. No en IRQ y sucede una IRQ de mayor prioridad, que si lo utiliza.
- El handler de menor prioridad no utiliza la FPU *antes* de la ocurrencia de la IRQ de mayor prioridad.
- Puede que utilice la FPU luego de que se atiende el handler de mayor prioridad, o no. Es irrelevante para el stacking.
- El handler de mayor prioridad en algún punto utiliza la FPU.

Lazy Stacking



Lazy Stacking

- **Ejemplo 5**: Se utiliza la FPU en software interrumpido, en IRQ y sucede una IRQ de mayor prioridad.
- Ambos stack frame generados (aplicación interrumpida e IRQ de menor prioridad) contienen registros FPU.
- FPCAR contiene punteros a ambos espacios reservados.

NOTA: *Ver imagen desde documento para mayor claridad, por cuestiones de espacio. Cortex-M4(F) Lazy Stacking and Context Switching, Application Note 298, Pag. 13*

Lazy stacking en OS

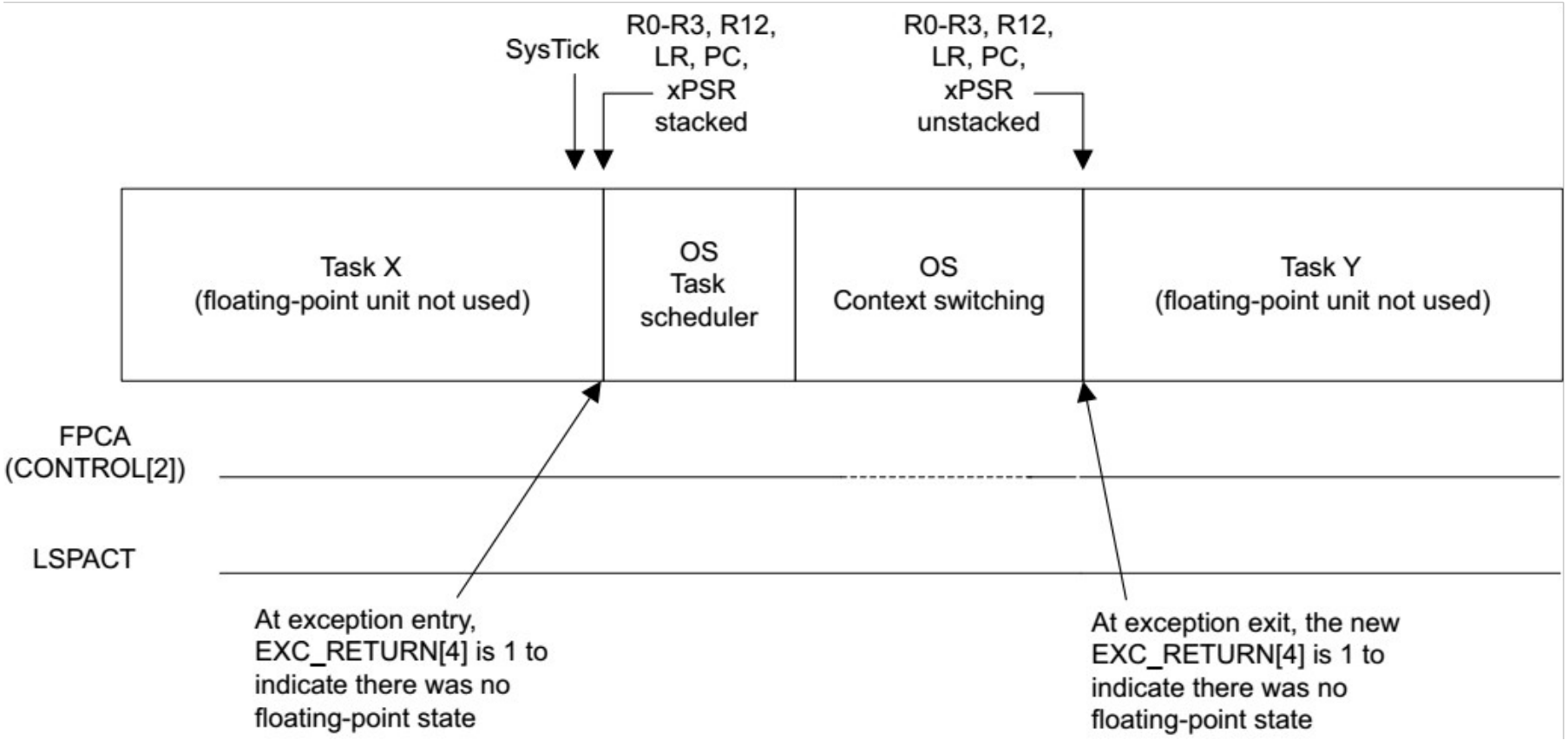
Lazy Stacking

- Es muy importante que el OS soporte la funcionalidad de lazy stacking.
- Basado en la utilización de la FPU por las tareas, existen 3 casos posibles:
  - Ninguna tarea utiliza la FPU.
  - Una sola tarea utiliza la FPU.
  - Mas de una tarea utiliza la FPU.

Lazy Stacking

- **Caso 1**: Ninguna tarea utiliza la FPU.
- FPCA = 0 antes de entrar a la excepción para el cambio de contexto.
- El punto anterior implica EXEC_RETURN[4] = 1 al entrar en la excepción.
- No hace falta guardar el contexto de la FPU (stack frame básico).

Lazy Stacking



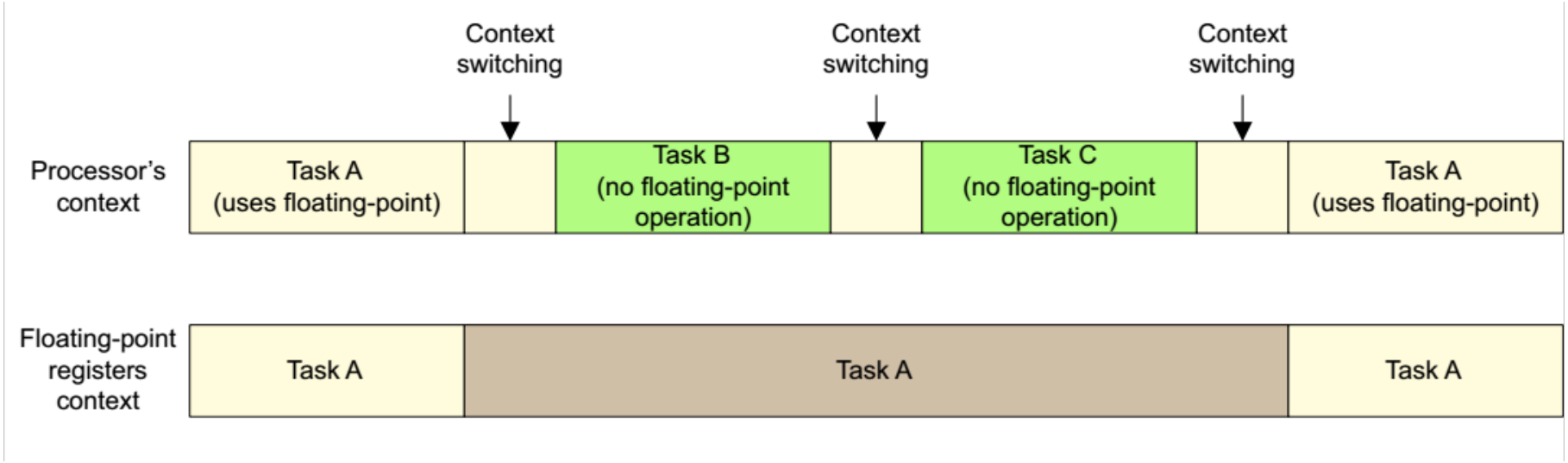
Lazy Stacking

- Para este caso, puede haber múltiples interrupciones con distintas prioridades que utilizan la FPU.
- Dado que se anidan los handlers, el mecanismo de lazy stacking se encarga de manejar esto.
- Los registros restantes de la FPU son manejados automáticamente por el compilador (igual que en baremetal).

Lazy Stacking

- **Caso 2**: Una sola tarea utiliza la FPU.
- Si es sabido que ninguna ISR utiliza la FPU, se puede evitar el stacking de los registros de FPU.
- Ahorro de espacio en stack.
- Es recomendable no evitarlo, de todas maneras el mecanismo de lazy stacking optimiza los tiempos de latencia.
- Lazy stacking está activado por defecto luego del reset.

Lazy Stacking



Lazy Stacking

- Existen técnicas para deshabilitar la FPU y ahorrar energía durante la ejecución de tareas que no la utilizan.
- Excesiva complejidad para las condiciones de trabajo de esta materia (EDU-CIAA alimentada desde puerto USB).
- Explicadas en detalle en Nota de aplicación 298 de ARM, página 19.

Lazy Stacking

- **Caso 3**: Más de una tarea utiliza la FPU.
- Para no complejizar el proceso, se propone el siguiente procedimiento:
  - FPU siempre activa.
  - Al entrar a PendSV, checkear EXCEC_RETURN[4] (almacenado en LR).
  - Si EXCEC_RETURN[4] = 0, se ejecuta PUSH de S16-S31
  - Si EXCEC_RETURN[4] = 1, no se ejecuta PUSH.
  - Push de registros adicionales de core y cambio de contexto.

Lazy Stacking

- Pop de registros adicionales de core y LR.
- Si EXCEC_RETURN[4] = 0, se ejecuta POP de S16-S31
- Si EXCEC_RETURN[4] = 1, no se ejecuta POP.
- BRANCH sobre el contenido de LR.

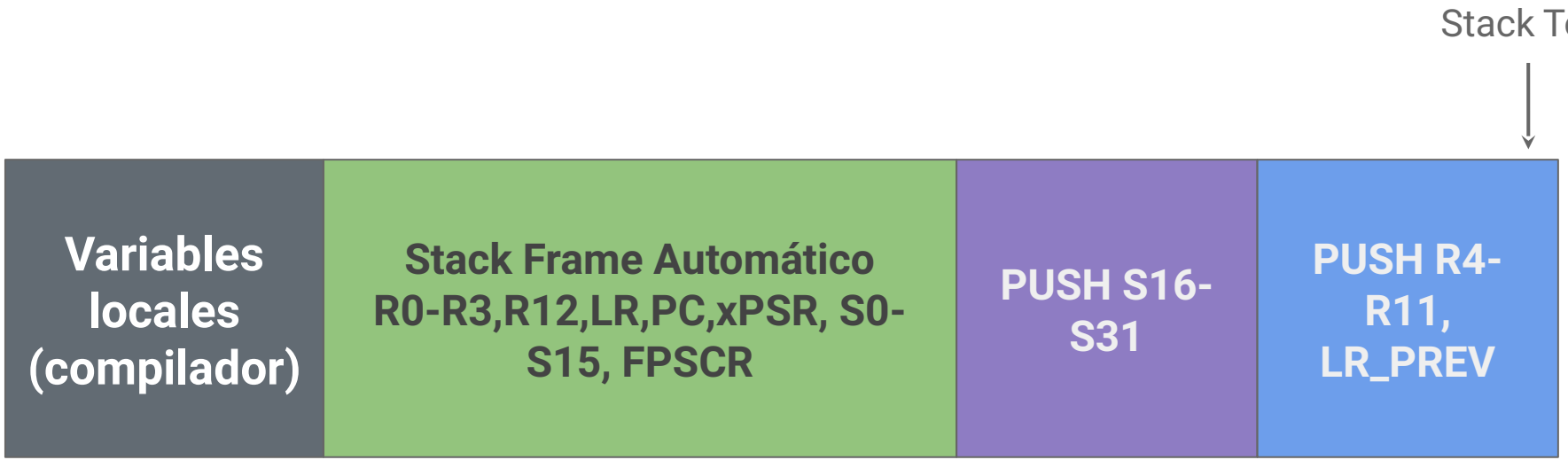
- El código resultante de este procedimiento se compone por 4 líneas adicionales de asm.
- El stack queda intercalado, pero a fines prácticos no tiene importancia.

Lineamientos

```
tst lr,0x10             //bit 4 del EXEC_RETURN es cero?
it eq
vpusheq {s16-s31}       //Si, PUSH de FPU regs

push {r4-r11,lr}
mrs r0,msp
bl getContextoSiguiente
msr msp,r0
pop {r4-r11,lr}

tst lr,0x10             //bit 4 del EXEC_RETURN es cero?
it eq
vpopeq {s16-s31}        //Si, POP de FPU regs

bx lr                   //retorno de PendSV
```

Lazy Stacking

- Conformación de stack para una tarea que utiliza la FPU luego de que se guarda su contexto; justo antes de llamar a getContextoSiguiente().



Lazy Stacking

HANDS ON

1. Implementar cambio de contexto con FPU

2. Implementar hooks restantes.

Gracias.