Sistema de Interrupciones

Sistemas con Microprocesadores

Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

http://www.microprocesadores.unt.edu.ar/procesadores

Cronograma

Actividad	Inicio	Descripción	Fin
Presentación	19/08	Reglamento de la Materia	✔
Tema 1	19/08	Estructura de las computadoras	✔
Tema 2	26/08	Proyecto con un microcontrolador	✔
Tema 3	30/08	Descripción funcional de microprocesador	✔
Tema 4	13/09	Programación en lenguaje ensablador	✔
Tema 5	25/09	Descripción general de un microcontrolador	✔
Tema 6	27/09	Estructura general de microcontrolador	✔
Parcial	09/10	Primer examen parcial	✔
Tema 7	14/10	Sistema de Interrupciones	←
Tema 8	21/10	Entradas y salidas digitales	
Tema 9	28/10	Entrada/salida con perifericos	
Tema 10	06/11	Temporizadores	
Proyectos	25/11	Seminarios de Proyectos	
Parcial	04/12	Segundo examen parcial	

NVIC – Bibliografía

- Capítulo 9 – User Manual UM10503
- Cortex M4 User Manual – cap 2.
- Yiu – Capítulos 7, 8, 9.
- ARMv7M – Architecture Reference Manual.
- Bajar ejemplos de nuestro sitio
  - Para correr en la placa
  - Probar en la práctica como funciona

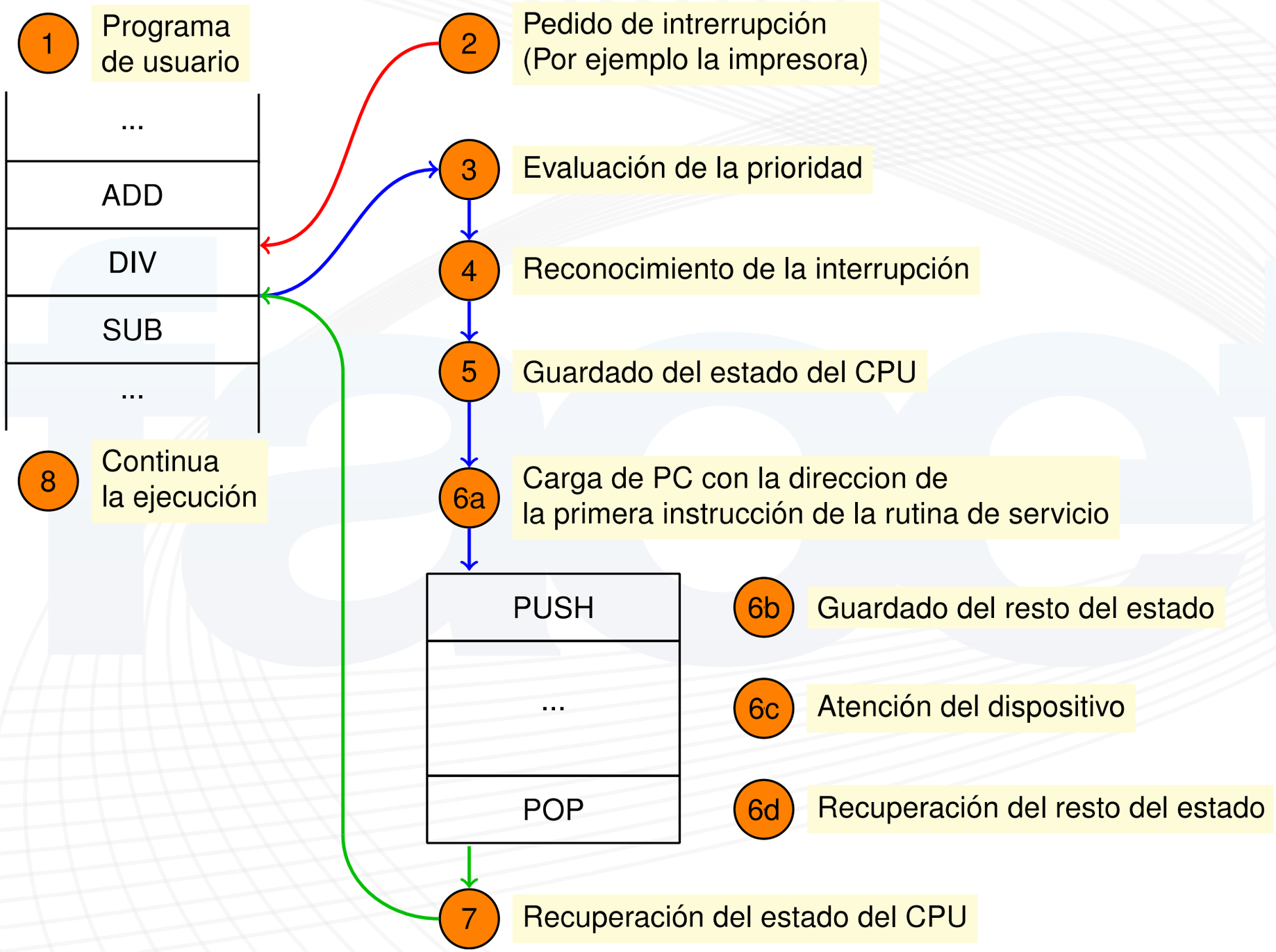
Interrupciones en la vida real

1) Un estudiante está estudiando (Ejecuta una tarea)
2) Lo llaman por teléfono (Pedido de Interrupción)
3) Decide si atenderá la llamada (Consulta de Prioridades)
4) Indica que sí atenderá (Reconocimiento de la Interrupción)
5) Pone un señalador en su libro (Guarda el estado de su tarea)
6) Atiende el teléfono (Sirve a la interrupción)
7) Abre el libro en el señalador (Recupera el estado de su tarea)
8) Retoma el estudio (Continua la tarea original)

Interrupciones en un procesador

1. El procesador ejecuta un programa (Ejecuta una tarea)
2. La impresora requiere nuevos datos para imprimir activa la señal /IRQ (Pedido de Interrupción)
3. Prioridad del Programa < Prioridad de la Interrupción (Consulta de Prioridades - suponemos que sí)
4. Señal de salida INTA (Reconocimiento de la Interrupción)
5. Guarda el valor actual del PC y ciertos registros (Guarda el estado de la tarea actual)
6. Atiende la interrupcion
   - Carga el PC con la dirección de la rutina de servicio (Sirve a la interrupción)
   - La rutina de servicio guarda el resto del estado al comenzar
   - Ejecuta los pasos necesarios para completar la nueva tarea
   - La rutina de servicio recupera el estado guardado al comienzo
7. Al terminar recupera el valor del PC y ciertos registros (estado de su tarea)
8. Continua la ejecución del programa (Continua la tarea original)

Punto de Vista del Programador



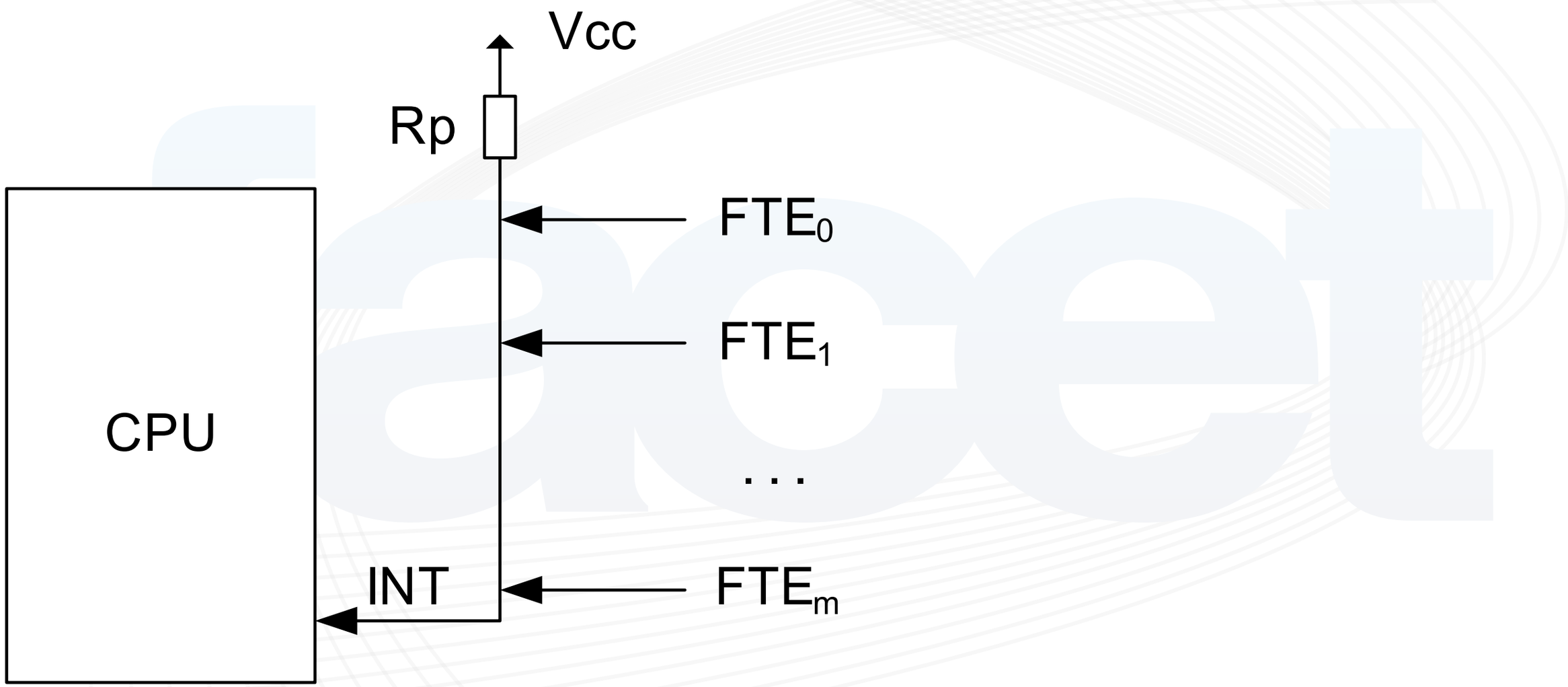
Características de las INT

- Interrupciones
  - Causada por eventos externos
  - Asíncronas con la ejecución del programa.
  - Reclaman un servicio, ISR (S.O. en general)
- Transparentes al Programa Principal
  - No tiene forma de saber cuándo llega.
  - No puede prepararse para ello.
  - ¿Cómo sabe que llegó una interrupción?

Múltiples Fuentes de Interrupción

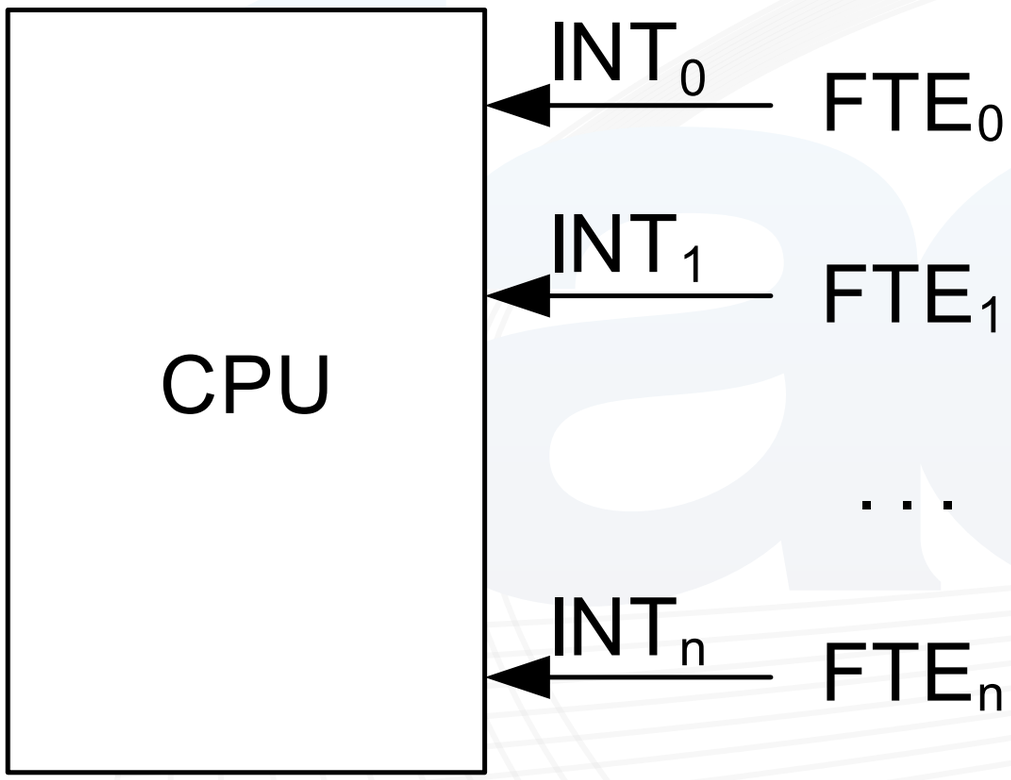
- Suena el Teléfono, el timbre y llaman a almorzar...
- Problemas
  - Conexión Externa.
    - ¿Cómo se conectan al CPU?
  - Identificación.
    - ¿Cómo se identifica quién pide Interrupción?
  - Simultáneas
    - ¿Qué pasa si se pide más de una a la vez?
  - Anidamiento
    - Mientras se sirve una Interrupción aparece otra.

1.1 Conexión: Colector Abierto.

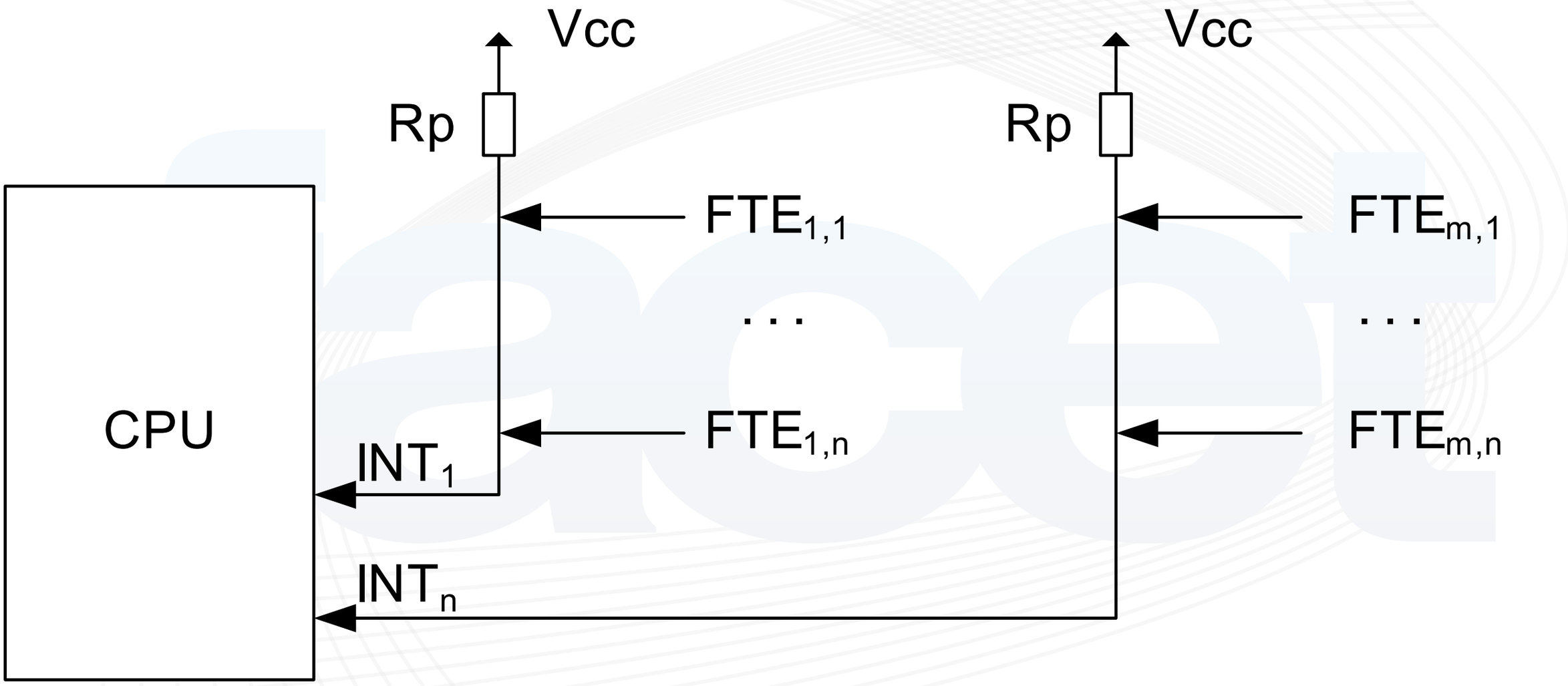


1.2 Conexión: Múltiples Líneas

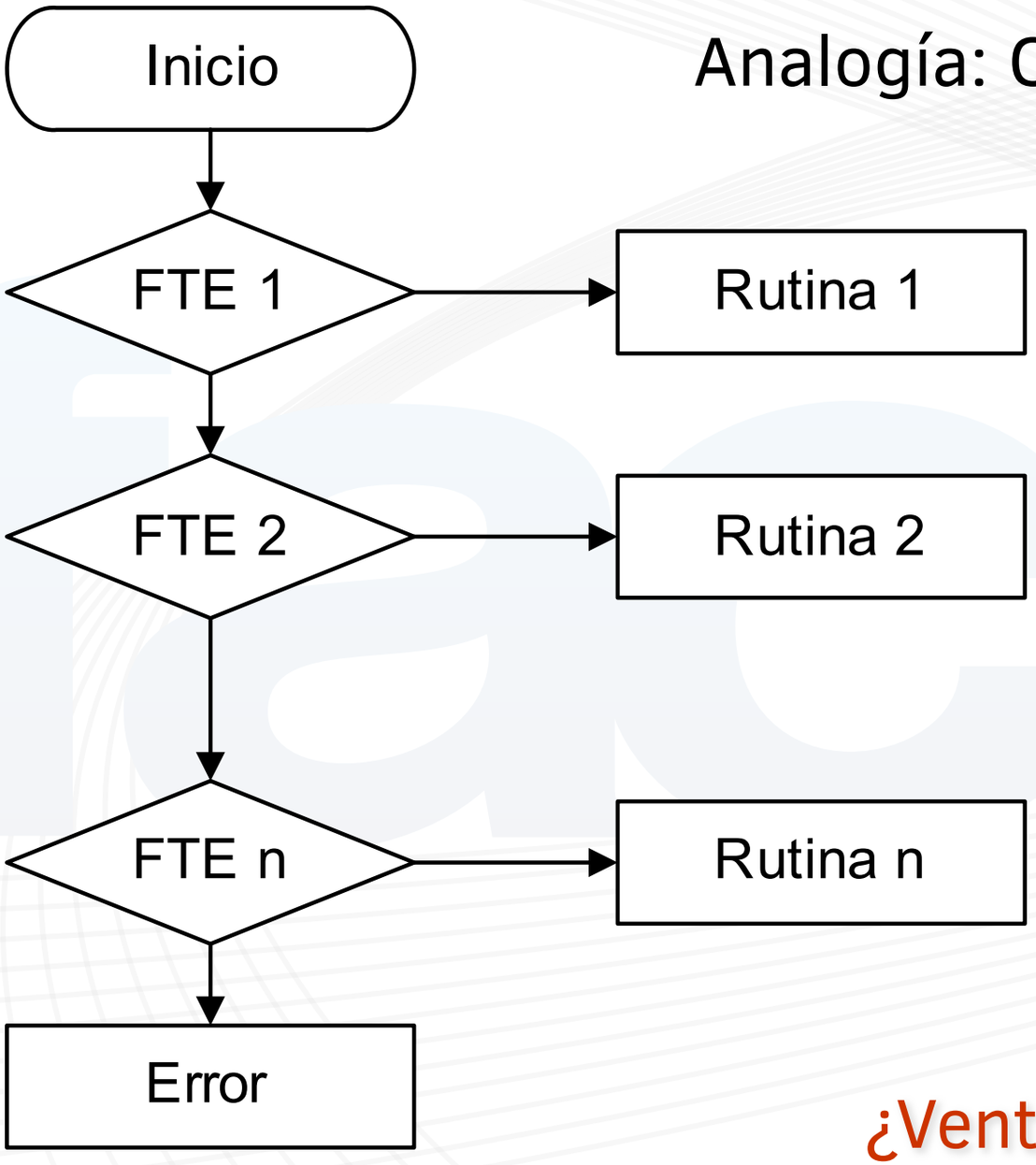
¿Ventajas y Desventajas?



1.3 Conexión: Esquema Mixto



2.1 ID por Software: Polling



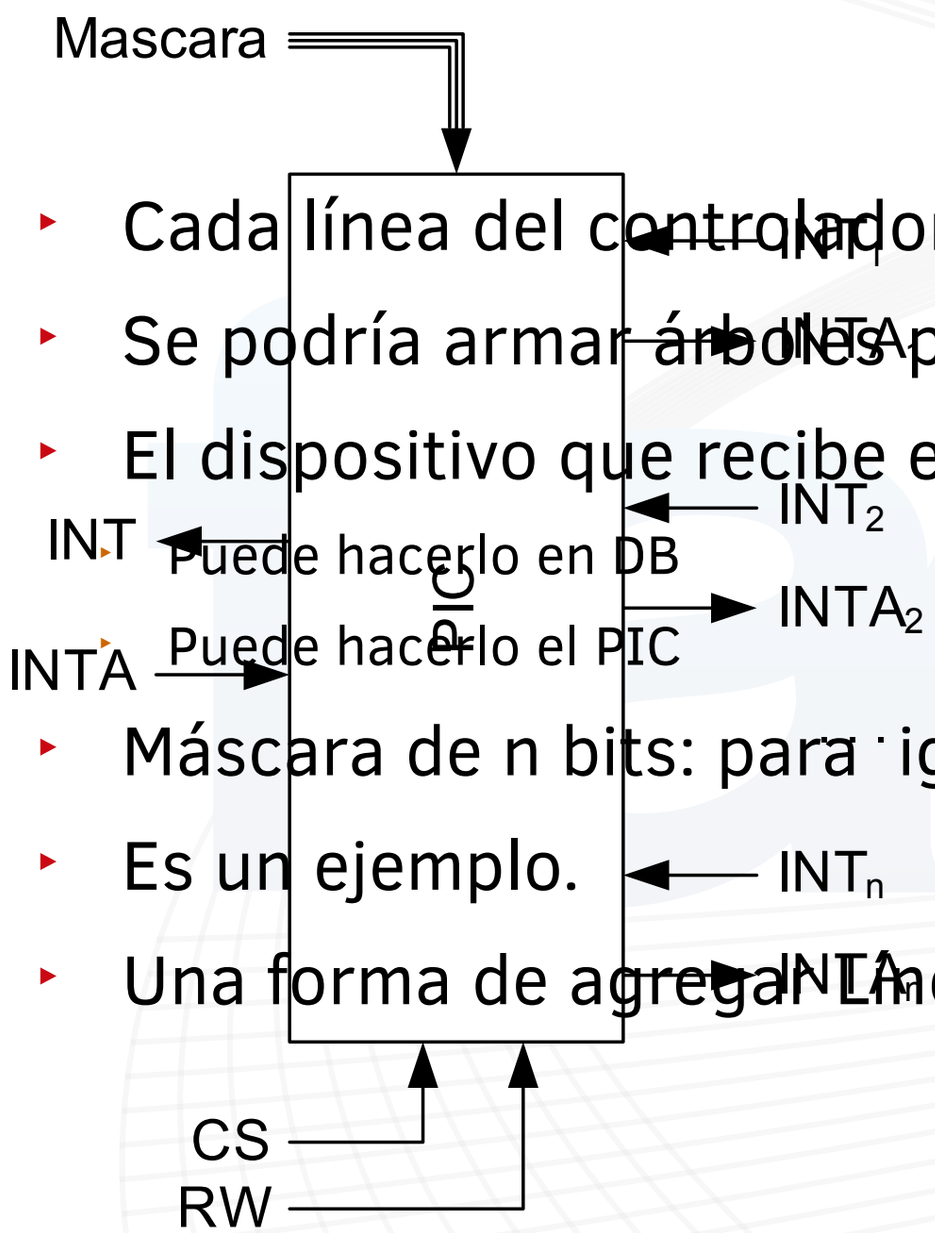
Analogía: Camarero de Hotel

¿Ventajas y Desventajas?

2.2 ID por Hw: Múltiples Líneas

- Analogía: Hotel con Tablero de Control.
- Alternativas de Implementación:
  - Cada línea se identifica con un número
  - Según el número de cada línea de interrupción, el CPU busca en una tabla la dirección de la Rutina de Interrupción (DIR).
  - Se llaman interrupciones vectorizadas.
- ¿Desventajas?

Control de Prioridad de Int. (PIC)



- Cada línea del controlador tiene una prioridad asociada
- Se podría armar árboles para extender num líneas
- El dispositivo que recibe el INTA debe identificarse
  - Puede hacerlo en DB
  - Puede hacerlo el PIC
- Máscara de n bits: para ignorar pedidos.
- Es un ejemplo.
- Una forma de agregar Lineas por Hw

PIC en Cortex-M4

- Se llama NVIC
  - Nested Vector Interrupt Priority.
- Maneja las interrupciones múltiples
  - Prioridades.
  - Vectores de Interrupción.
- Es más complejo, maneja más funciones.
- Pero… si tengo multiples fuentes ¿cómo las enmascaro independientemente?
  - EL NVIC permite enmascarar cada dispositivo interno que pide interrupciones.
  - Veremos más detalle próximamente.

Interrupciones en ARM-Cortex- M

- Puede haber varias fuentes capaces de ocasionar interrupciones.
- Cada fuente de interrupción tiene un bit de “**armado**” o habilitación
  - En “1” habilita (**arma**) a los dispositivos para que **puedan disparar** un pedido de INT.
  - En “0” se pone a aquellos que no se permite que **disparen pedidos**.

Interrupciones en ARM-Cortex- M

- Cada fuente de interrupción, Hw, cuenta con un flag de “disparo” o pedido.
- Cuando el dispositivo pide interrupción y está armado:
  - Hw pone flag=1
  - Hubo un disparo y está pendiente de respuesta.

Interrupciones en ARM-Cortex- M

- Cada dispositivo tiene definida una prioridad, que se evalúa cuando pide interrupción.
  - Las prioridades se incrementan en sentido descendente.
- La interrupción tiene que estar **habilitada** en el CPU.
  - Global interrupt enable bit, I=0, en el registro **PRIMASK**
  - Prioridad de la INT > prioridad Programa (**BASEPRI**).

Condiciones de Interrupción - I

- deben cumplirse cuatro condiciones necesarias simultáneamente:
  - Habilitado en el CPU: I=0 in PRIMASK.
  - Armado el dispositivo: el bit del Hw=1
  - Nivel Prioridad: Nivel de IRQ “menor” que BASEPRI
  - Pedido: Una acción externa de Hw setea la bandera de pedido.

Condiciones de Interrupción - II

- La interrupción queda pendiente si hay disparo pero cualquier otra condición no se cumple.
  - Se atiende solo si todas las condiciones se cumplen.
  - Necesario reconocer (acknowledge) la INT.
    - Resetear bit de disparo en ISR, sino se interrumpirá de nuevo cada vez que se retorne, ilimitadamente. **¿No se autointerrumpe?**
    - Lo hace Hw externo capaz de detectar que ISR comienza a ejecutarse.

Procesamiento de la INT - ISR

- El Hw realiza lo siguiente:
  1. Esperar fin de la instrucción actual.
  2. Suspender ejecución y push de 8 registros (R0-R3, R12, LR, PC, PSR) en stack.
  3. LR=0xFFFFFFF9 (indica que hubo una Interrupción)
  4. Escribir en PSR el número de interrupción.
  5. PC ← Dirección inicial de ISR.

Ejecución de la ISR

1. Acknowledge: Pone a cero el flag que disparó la interrupción.
2. Salva registros no guardados automáticamente si los usa, en Stack.
3. Se comunica con variables globales **¿con quién? ¿por qué?**
4. Servicio a la interrupción.
5. Restaura Registros del punto (2)
6. Se retorna del Prog. Principal con BX LR
   - Realiza Pull de los 8 registros del stack

Implementación en CORTEX-M4

- Si llega señal con prioridad=INT
  - No puede interrumpir ¿por qué?
- Si dispositivo la dispara de nuevo, luego de ACK, pone un pendiente.
- Cada interrupción se identifica con un número ID.
- El CPU busca la dirección de la rutina de interrupción en una palabra de M, diferente para cada ID, “vector de la interrupción”. Carga este valor en PC.
- Al retornar, la prioridad es la que había antes.

Pregunta Importante

- ¿Qué diferencias hay entre interrupciones y subrutinas?
- Las interrupciones son ASINCRONICAS.
- ¿Cómo sabe en donde está la rutina de interrupción?
- Mediante el Num de INT accede a una tabla de direcciones
  - Vector Int. Table.

3. Simultáneas.

- Son simultáneas las que ocurren en el intervalo de una instrucción
  - En ARM-M4 hasta el apilado de registros por Hw (al final se chequea).
- También son simultáneas las que quedaron sin atender de una ronda previa.
- Se resuelve por prioridades.
- El sistema de prioridades debería ser equitativo.
  - Equitativo: que nadie quede sin atender.

Simultáneas - Prioridades

- SW - Polling.
  - El que se encuesta primero es prioritario.
  - Permite esquemas elaborados más equitativos.
    - Ej. Round Robin con una y dos prioridades.
- Hw.
  - Cada línea tiene su prioridad.
  - Se las puede configurar por Sw.
  - También permite esquemas más equitativos.

4. Anidamiento

Hablo por teléfono y suena el timbre…

1. Define el Diseñador.
   - Usar solo en caso necesario, es complejo.
   - En el M4 se las puede enmascarar **¿cómo?**.
2. Si Admite, entonces
   i. El Hw permite solo con mayor prioridad.
   ii. El sistema Vectorizado lo permite sin problema.

Excepciones

- Interrupciones
  - Causada por eventos externos
  - Asíncronas con la ejecución del programa.
  - Reclaman un servicio (S.O. en general)
  - Se ejecutan entre instrucciones del programa.
  - Simple: suspenda y continúe el programa del usuario.
- Traps
  - Causados por eventos internos.
    - Condiciones excepcionales (OP-Code inválido)
    - Errores en Hw (parity)
    - Otros.
  - Algunas son sincrónicas con la ejecución del programa.
  - Se debe abortar la instrucción.
  - El problema debe ser remediado por SO – si se puede.
  - Si no se puede, se aborta el programa
  - RESET si falla generalizada que no puede resolverse.

Modos: Usuario / Supervisor

- Nivel de Privilegio: Supervisor/Usuario
- En modo Supervisor
  - Todas las instrucciones permitidas.
  - Todos los dispositivos y áreas de M.
- En modo Usuario
  - No se permite acceder a los dispositivos.
  - No se permite acceder a algunas áreas.
  - Para asegurar que un usuario no provoque problemas. ¿Cuáles?

Modos: Thread/Handler

- Implementación ARM-M4
- Modo Thread
  - Cuando un programa está corriendo.
- Modo Handler
  - Cuando una interrupción se ejecuta.

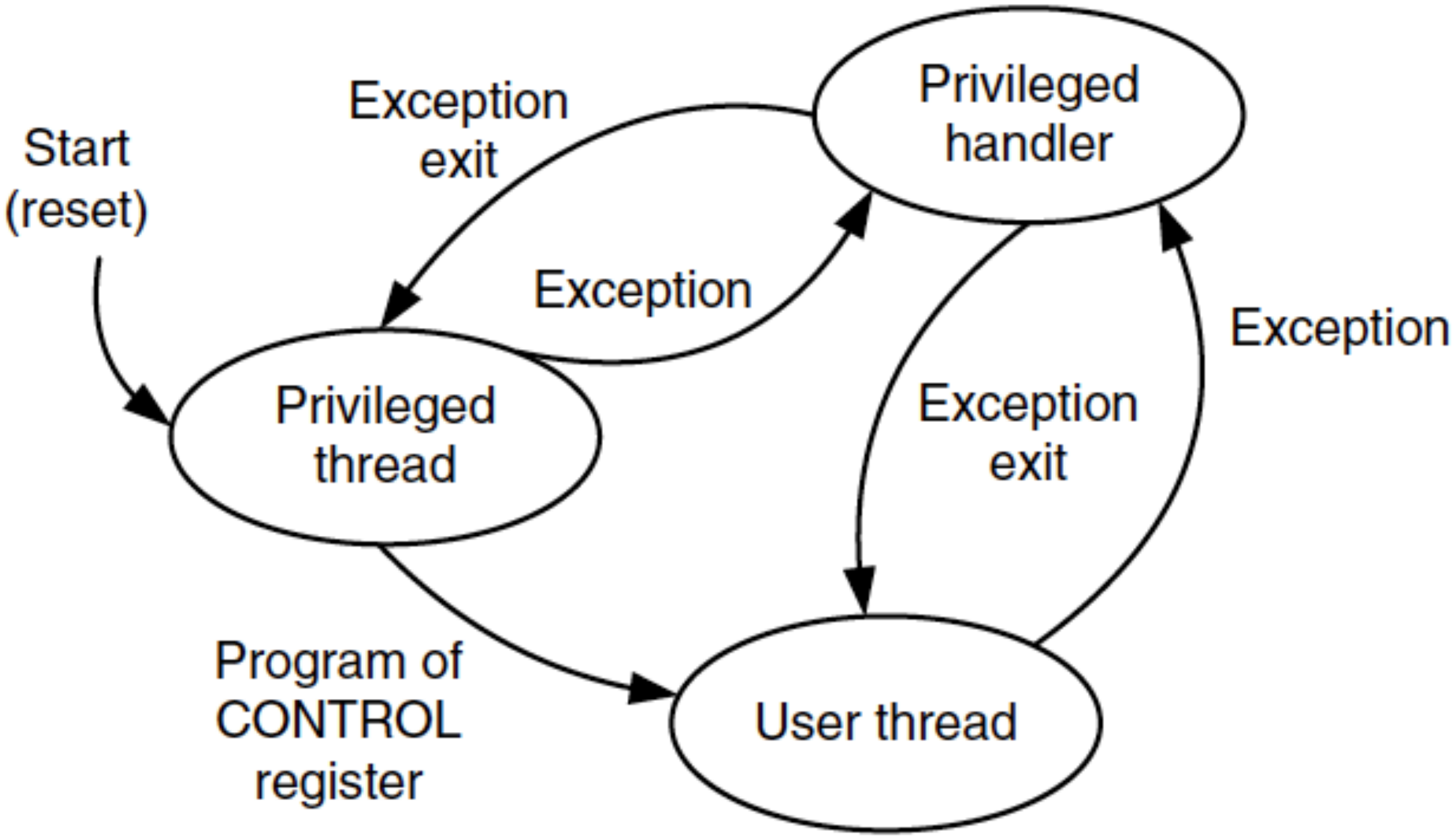
Implementación Cortex-M4

- Las ISR siempre en nivel supervisor. Se les llama “handler” porque en general manejan un dispositivo (modo handler).
- El programa puede estar en nivel Supervisor o Usuario.
  - En general en nivel usuario.
    - En este caso tiene un stack distinto PSP (puntero)
    - El supervisor tiene MSP (puntero). ¿Por qué?
  - ¿Cómo corre el SO?
    - En nivel supervisor.

Modos del Procesador

- Solo en nivel Supervisor.
- Modo Supervisor Thread.
  - Corre un programa, por ej. SO.
  - Así arranca con el reset.
  - Trabajaremos en este modo, ya que somos desarrolladores.
- Modo Handler.
  - Para respuesta a Excepciones.
    - Interrupción se considera como excepción.

Operación: Modos y Privilegio



Supervisor Call (SVC)

- Similar a una Interrupción, pero provocada por una Instrucción.
- En realidad es un Trap.
- Salta a una rutina de Excepción, de acuerdo a un vector propio.
- Se usa para pedir servicios al Sistema Operativo desde “thread usuario”.

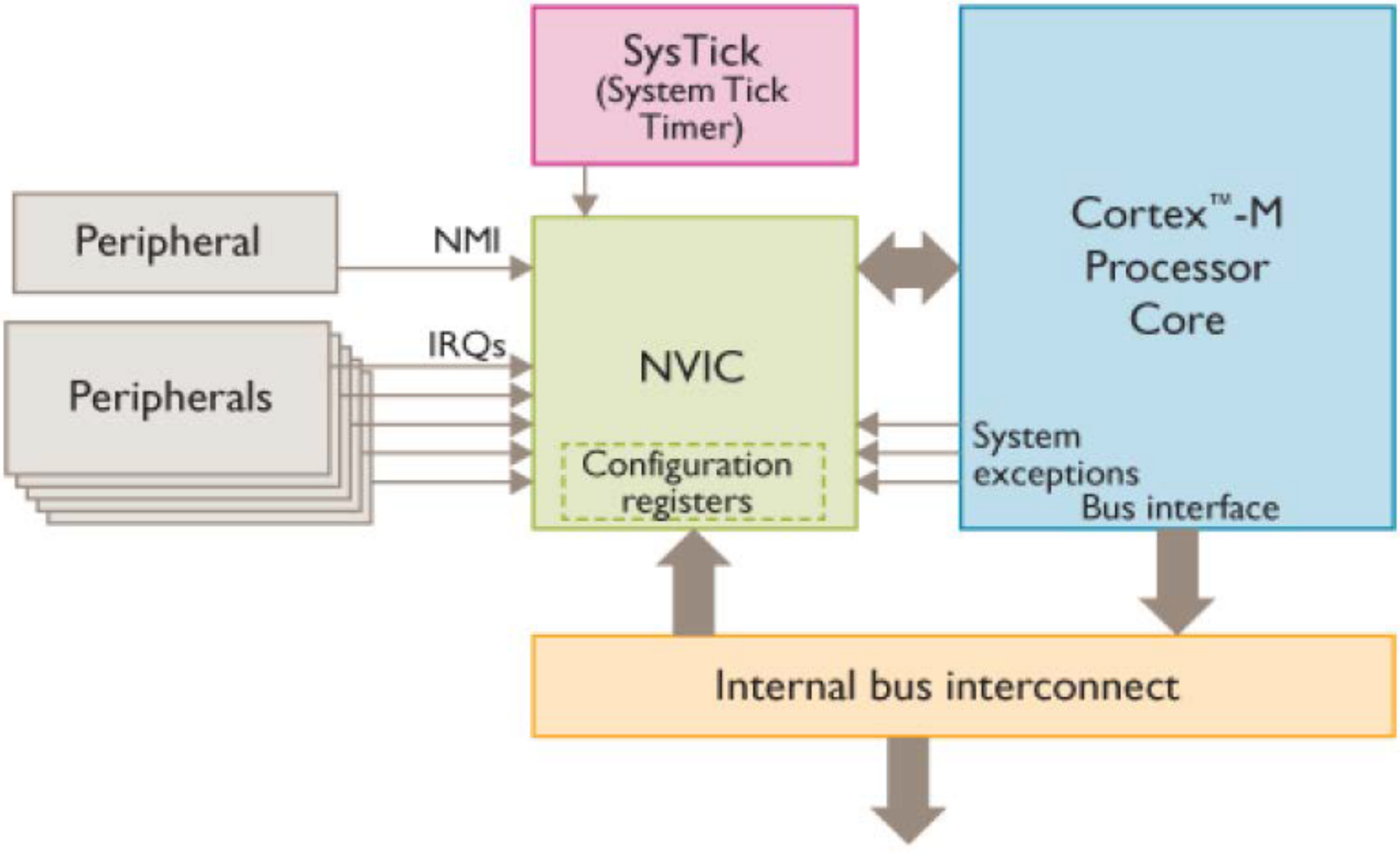
Reset

- ¿Qué pasa cuando se da tensión al µC?
- Problemas a Solucionar:
  - Registros y RAM indeterminados.
  - ¡Por tanto PC indeterminado!
  - Interfases E/S indeterminados.
- Es necesario asegurar un valor inicial al PC, SP, puertos y dispositivos µC.
- Circuito de espera por el transitorio.
- Que PC apunte a una Rutina de booteo.
- Se lo trata como una excepción de máxima prioridad.

Implementación ARM.

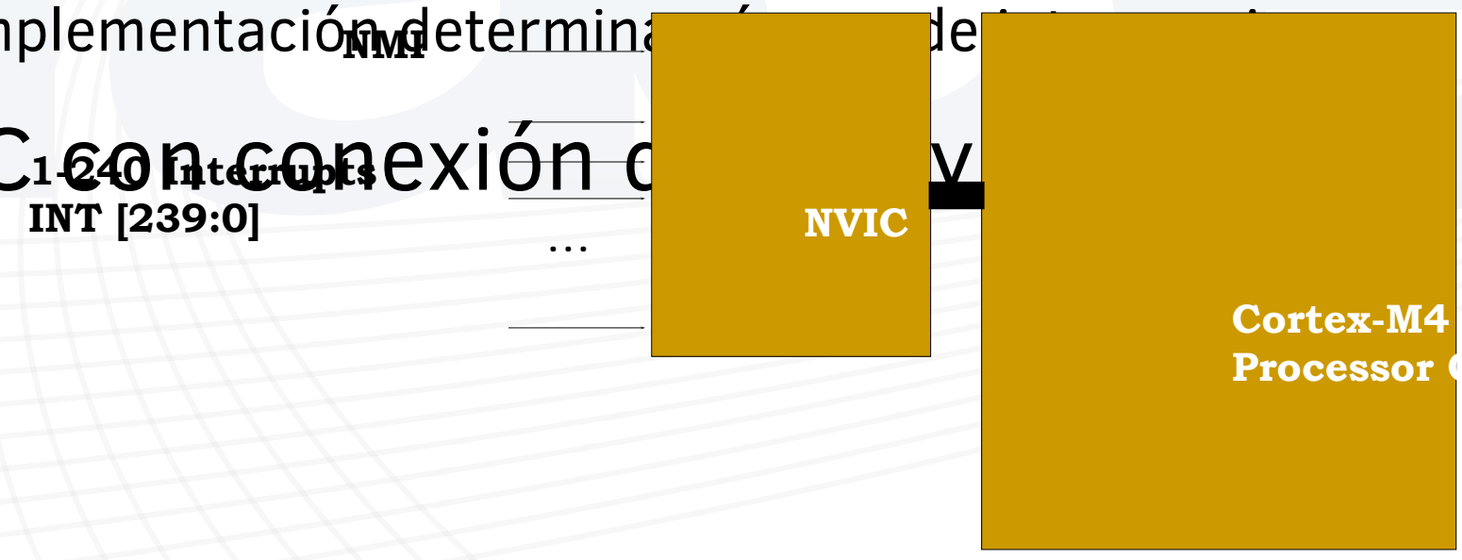
- Tabla vectores y programa de booteo en FLASH.
  - Vector 0: valor inicial MSP
  - Vector 1: valor inicial PC (Reset)
  - Reset se considera como Excepción #1
- En aplicaciones embebidas
  - En general Programa y constantes en Flash.
  - Variables en RAM.
  - ¿Dónde están los dispositivos E/S?

Implementación Cortex-M4



Implementación Cortex-M4

- Una Interrupción no enmascarable (NMI)
- Hasta 240 interrupciones priorizables
  - Pueden ser enmascaradas
  - La implementación determina
- NVIC con conexión d v a CPU.



M4: Excepciones del Núcleo

- **Tres con Prioridades Fijas, el resto configurable**
  - **Reset:** varias causas (POR, BOD, …), prioridad=-3
  - **NMI:** Solo Reset tiene más prioridad. Prioridad=-2
  - **Hard Fault:** Falla de Hw no especificada. Prioridad=-1
- **Faults – además de Hard Fault:**
  - **Memory Management**: Violación de Protección de M (MPU)
  - **Bus Fault**: Falla en el Bus.
  - **Usage Fault**: Falla en programa: instrucción no definida, división por cero, etc.
- **SVCall:** Pedidos del programa al SO, serv. privilegiados.
- **SysTick Interrupt:** Pedida por un timer interno del núcleo del ARM. Veremos más adelante.

ARM – Registros Especiales

Figure 3. APSR, IPSR and EPSR bit assignments

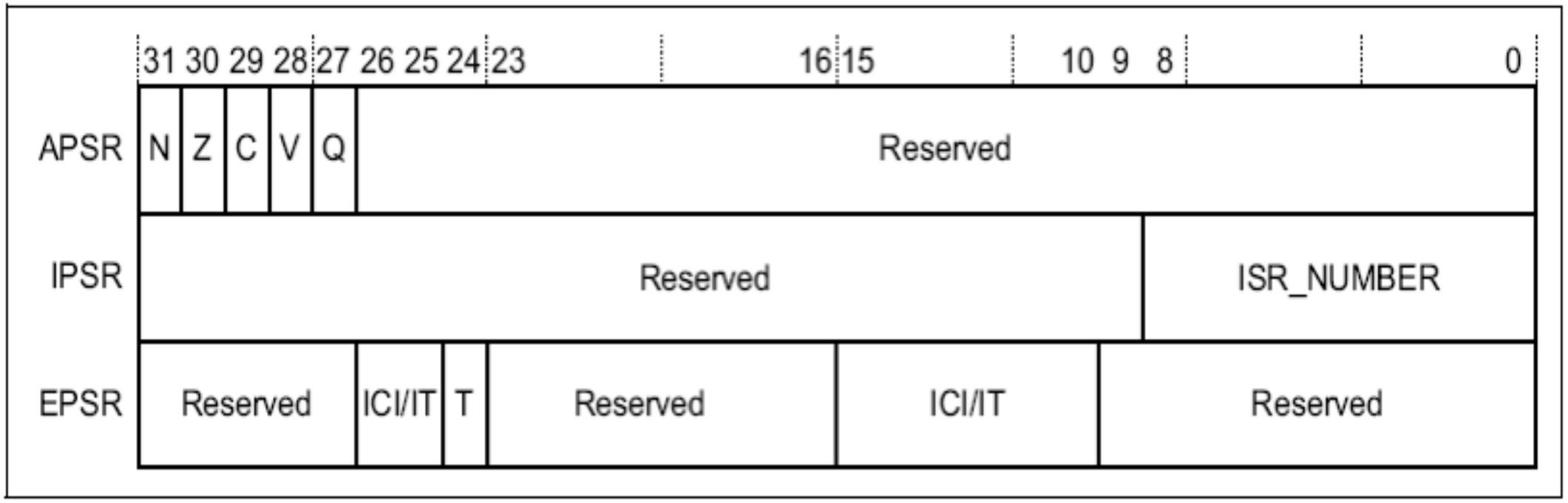
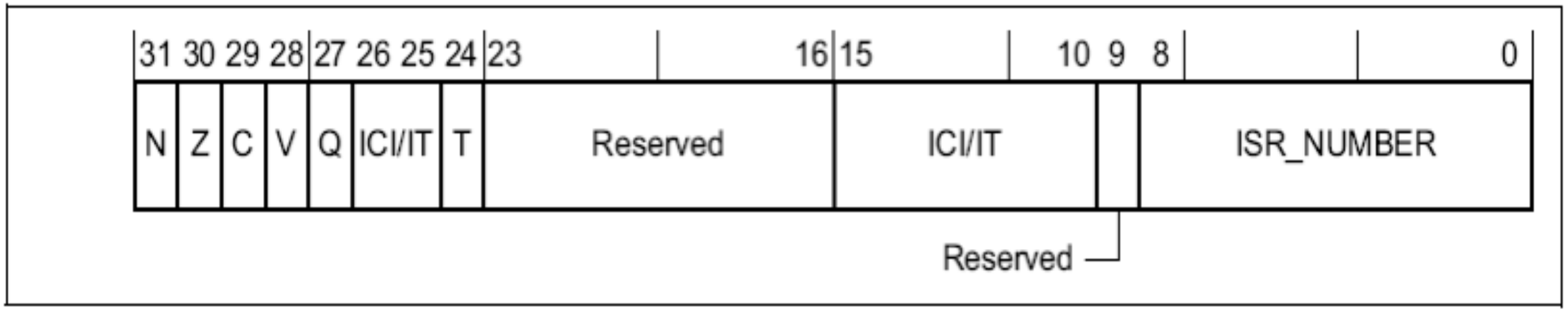


Figure 4. PSR bit assignments



Interrupt Program Status Register

Bits	Description
Bits 31:9	Reserved
Bits 8:0	**ISR_NUMBER:** This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0(1) ....

Figure 2-3, The IPSR Register.

Registros Especiales del CPU

- PRIMASK
  - Solo tiene bit I, en lsb.
  - CPSIE I → habilita interrupciones (I=0)
  - CPSID I → inhabilita interrup. (I=1)
  - Es para todas las excepciones excepto las de errores de Hw, NMI y Reset.
- FAULTMASK
  - Similar a PRIMASK, un solo bit.
  - Inhabilita todas menos NMI y Reset.
  - CPSIE F – CPSID F. (F=0, F=1, respectiv.)
  - Supondremos siempre en 0.

Registro BASEPRI

- Indica la “menor prioridad” no admitida
  - BASEPRI = 0, no hace nada, cualquier prioridad de Int. se ejecuta – Default. BASEPRI > 0, INT se ejecuta solo si Prioridad(Int) < BASEPRI
- BASEPRI setea la prioridad de las interrupciones permitidas, (siempre ≥ 0).
  - Ej: si BASEPRI = 3, interrupciones con prioridad
    - -3 a 2 pueden ocurrir, suspendiendo esta interrupción.
    - 3 a 7 serán pospuestas hasta que termine la nueva ISR.

Instrucciones Reg. Especiales

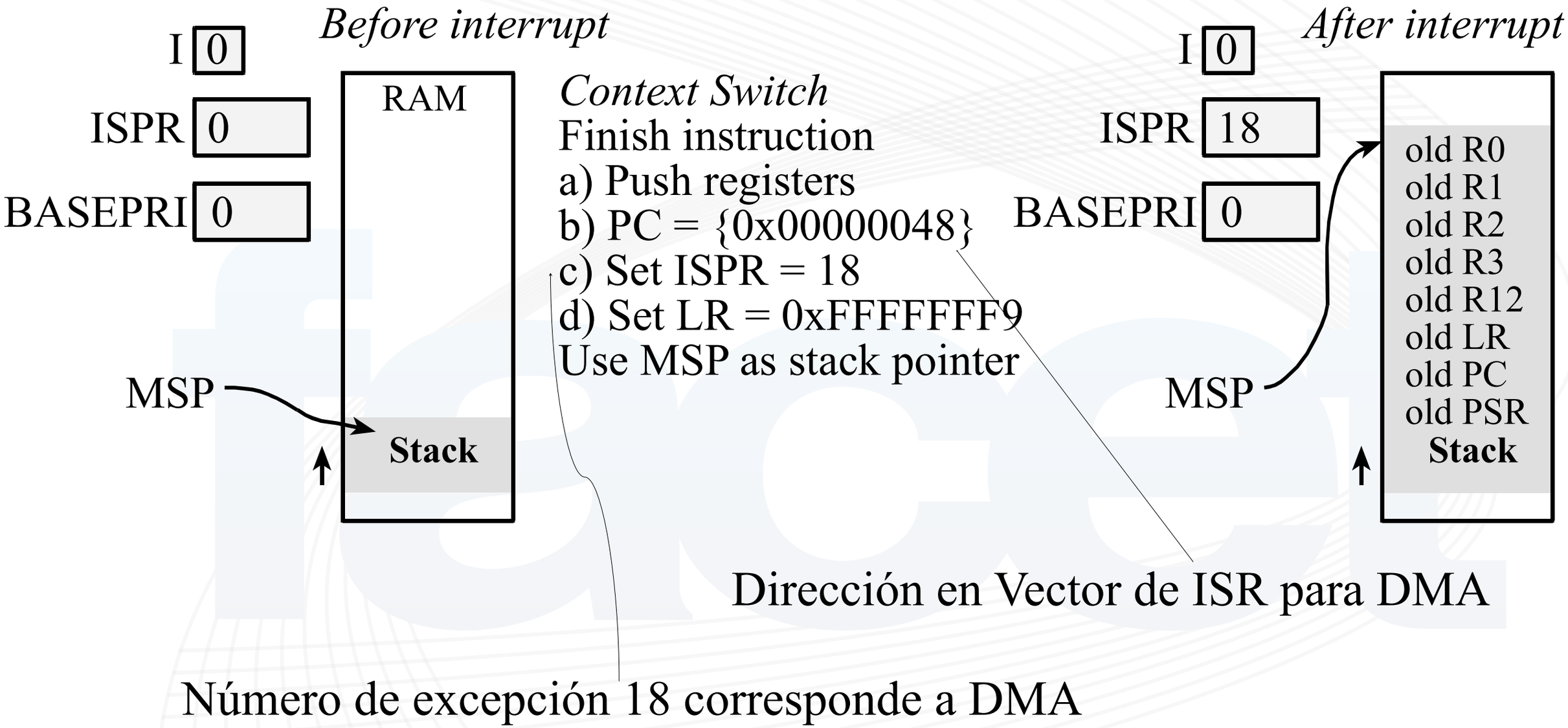
- Para registros especiales (R_esp)

```
MSR R_esp,Rn ; R_esp ← Rn
MRS R_esp,Rn : Rn ← R_esp
```

- Para cambiar BASEPRI

```
MOV R0,#prioridad
MSR BASEPRI,R0
```

Ej: Cambio de Contexto (INT#18)



Tipos de Excepciones

- ARM diferencia 2 tipos de excepciones
- Internas.
  - Causadas por núcleo ARM
  - Son 15
    - Se cuenta RESET como #1
    - Se cuenta SysTick
    - Especificado por ARM y viene con el núcleo.
    - Numeradas desde 1 a 15.
- Externas (fuera del núcleo).
- Núm de excepción: se cuentan todas.
- Núm Int. Externa: IRQ #0 = Exc #16.

NVIC - Visión Lógica.

- Hasta 255 excepciones.
  - Hasta 240 interrupciones.
  - En LPC43xx hasta 53 interrupciones.
- Cada excepción se identifica con un #
- # Interrupción Ext. = # Excepción – 16.
- Se accede a una tabla lógica.

Campos de la tabla

- **Vector** de la Interrupción, 32 bits. (por #Excepción)
- **IP**: 8 bits - fija prioridad INT (por #Interrupción, en adelante).
- **ISE:** 1 – habilita bit interrupción (Set Enable Interrupt).
- **ICE**: 1 – inhabilita bit interrupción. (Clear Enable Interr.)
- ISE e ICE: solo se puede escribir 1. ICE=1 pone IM=0 y ISE=1 lo pone en 1. (IM: interrupt enable mask pero solo para esa INT)
- **ISP**: 1 – pendiente de ejecución (Interrupt Set Pending)
- **ICP**: 1 – para quitar pendiente por programa (Int Clear Pending).
- En cualquiera de los anteriores, una lectura muestra el estado.
- **IAB** 1 – INT activa en ejecución. Solo lectura. (IA Bit)

NVIC – Visión Física I.

- Varias tablas separadas por función.
  - Ordenadas por número de interrupción.
  - A partir de direcciones iniciales.
    - Se acceden en gral en función de #interrupt.
    - Se ubican en Memoria.
- Tabla de Vectores – al principio de la memoria.
  - Única por número de **excepción interna.**
  - Se puede cambiar su ubicación para RAM cuando se hace debugging, por ejemplo.

NVIC IRQ En/Disable Registros

- Registros para habilitar interr. (NXP LPC43xx)
  - Nombre **ISER_0 e ISER_1**. Int Set Enable Reg. 0 (1).
  - Cada reg de 32 bits tiene un bit para cada # int.
  - Son dos registros ¿hasta cuántos dispositivos?
  - Escribiendo un 1 se habilita la interr.
  - Un 0 no tiene efecto.
- Registros para inhabilitar interr
  - Similares a los anteriores. **ICER_0 (1)**.
  - Escribiendo un 1 se inhabilita la interr.
  - Ej: ICER_0[i]=1 → IM[i]=0 automático.
  - Un 0 no tiene efecto. ¿Por qué se diseña así?

NVIC Niveles de prioridades

- Prioridades numéricas menores son “más altas”.
- Se define el concepto de grupo y subgrupo de prioridad.
- Se permite el anidamiento de excepciones con diferentes grupos de prioridad
- El subgrupo permite definir entre interrupciones simultáneas del mismo grupo.
- Si dos interrupciones tienen el mismo grupo y subgrupo entonces se define por el numero de excepción.

NVIC Niveles de prioridades

- La prioridad se configura con un registro de ocho bits que se puede partir en dos campos:
  - La parte más significativa define el grupo de prioridad.
  - El resto define el subgrupo de prioridad.
- La división en grupo y subgrupo se puede configurar
  - Se pueden asignar de 0 a 7 bits para el grupo
  - El resto corresponde al subgrupo
  - Máximo 128 niveles de grupo y hasta 256 niveles de subgrupo.
- Los registros de prioridad se pueden fabricar con menos bits
  - El fabricante puede usar de 3 a 8 bits para cada registro
  - El procesador NXP4337 tiene implementado solo 3 bits
  - En este caso los bits no implementados se consideran siempre como cero.

Registros de Prioridad

- **IPRx** (x=0…) – Interrupt Priority Registers x.
  - Divididos en campos de 8 bits.
  - Cada campo determina una prioridad.
    - 0 es la mayor.
  - Num reg= x = num IRQ/4
  - Byte en el reg = IRQ Mod(4)
  - Se emplean los bits más significativos del campo.
  - El resto se puede configurar como subnivel

Ejemplo:

- Calcular Núm x de registro y byte o bit para IRQ 45. Set PRTY(45)=4
- IPRx (campos de 1Byte, 4 por registro)
  - 45/4=11, resto=1 → IPR11, Byte=1
  - IPR11[17:15] = 4 (binario 100).
- ISERx e ICERx
  - 45/32=1, resto=13 → x=1, bit=resto=13.
  - Acceder a bit 13 de ISER1 para habilitar INT.
  - Bit 13 de ICER1 para inhabilitar INT.
- Idem para otros registros.

NVIC: pendientes y activos

- Una interrupción puede anidar a otra.
- Si no puede, pone bit pendiente=1 en un registro que vale para 32 IRQs.
- Cuando se ejecuta, pone pendiente=0, activo=1.
- Si anida, la que se interrumpió sigue con su bit activo en 1.
- Ojo. Si I=0 o no habilitada en NVIC, y llega el pedido, Pend=1!!

¿Cómo provocar INT por Sw?

- Poner “1” en bit **ISPRn[b]**
  - correspondiente al # de int.
- Si esa INT está habilitada en NVIC, I=0 y BASEPRI>PRI(Int):
  - Inicia INT al terminar la instrucción.
- Puede ser necesario generar INT por Sw.
- También se puede quitar por Sw una interrupción pendiente: **ICPRn[b]=0.**
- **IABRn[b]**, se lee para saber si está activa una interrupción.
  - Interrupt Active Bit Register (0 o 1).

Registros NVIC

Dirección	Nombre	Acc	Descripción
E000E100/4	ISER0/1	RW	1=habilita INT. Lectura: estado.
…180/184	ICER0/1	RW	1=inhabilita INT. Lectura: estado
…200/204	ISPR0/1	RW	1=fija INT pendiente. Lectura: estado
…280/284	ICPR0/1	RW	1=quita INT pendiente. Lectura: estado
…300/304	IABR0/1	RO	Lectura: INT activa.
…400/404	IPR0/1	RW	Asigna prioridad INT #1 a #8
…408/40C	IPR2/3	RW	Asigna prioridad INT #9 a #16
…410/414	IPR4/5	RW	Asigna prioridad INT #17 a #24
…418/41C	IPR6/7	RW	Asigna prioridad INT #25 a #32
…F00	STIR	WO	Genera SVC (supervisor Call)*

Ritual de Inicialización

- Son instrucciones que se ejecutan al inicializar el MPU, por única vez, para configurar interrupciones, puertos, etc.
- Inicializar estructuras de datos (punteros, colas…)
- Poner a cero el registro de pendientes (por si acaso)
  - ICPRn[b]=1
- Configurar NVIC
  - Escribir prioridades. (IPRn)Habilitar Int. (ISER)
  - Habilitar Int. (ISER)
- Armar interrupción en periférico.
  - Resetear Flag de disparo
  - Habilitar interrupción del periférico.
- Habilitar Interrupciones (si no lo estaban).
  - Si hace falta, configurar BASEPRI.
  - I=0 (CPSIE I)

Detalle Anidamiento

- Para retornar de ISR siempre BX LR
- LR se carga automáticamente
  - FFFFFFF9 desde Supervisor Thread.
  - FFFFFFF1 desde Handler.
  - FFFFFFFD desde USER Thread – PSP
- Dos registros de pila, intercambiables
  - PSP – pila para usuario.
  - MSP – pila para supervisor.
  - Trabajaremos desde Supervisor Thread **¿por qué?**

Respuesta a pedido INT

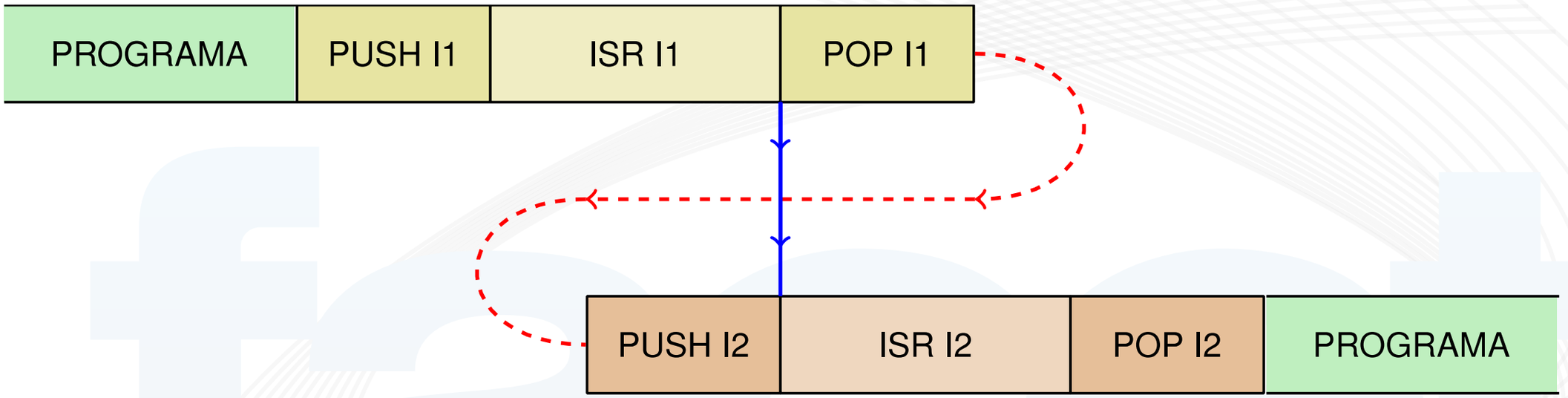
- Cuando responde, si el pedido no cambia a 0 antes de retornar de su ISR, se genera un pedido pendiente.
- Si aparece un flanco nuevo una vez que se atiende y reconoce la interrupción, se genera un pendiente.
- Una interrupción por flanco o por pulso debe permanecer al menos 1T para ser reconocida.

Respuesta rápida a Simultáneas

- Si cuando se responde una interrupción
  - y mientras se está apilando los registros
    - llega nueva interrupción más prioritaria.
- No reapilar, poner dirección ISR prioritaria en ISR.
  - Dejar viejo pedido como pendiente.

Retorno rápido a INT pendiente

- “Encadenamientos de Cola” al retornar



- Se ejecuta una interrupción I1 y hay otra I2 pendiente.
  - I1 no realiza POP en su retorno.
  - I2 no realiza PUSH en su entrada..
- Latencia baja cuando se pasa a otra interrupción pendiente.
  - Así funciona en ARM M4 – no necesariamente en otros proc.

Esperar tiempo fijo

- Hasta aquí lo hacíamos con lazos.
- Presentaremos un dispositivo que viene en el núcleo del ARM: SysTick Timer.
- Señala cuando se cumple un lapso de tiempo, por ejemplo 0,1 seg.
- Espera activa de 1 seg: esperar 10 señales.
- Espera inactiva: por interrupciones.

SysTick Timer

- Contador de 24 bits.
- *Decrementa* a la frecuencia del bus.
  - Ej: empleando clock interno de 48 MHz, se decrementa cada 1/48MHZ=20,8 ns
- Se parte de n hasta llegar a 0
  - Cuando llega a 0 se carga con n de nuevo.
  - Provoca una señal cada n+1 pulsos:
    - **next_value = (current_value-1) mod (n+1)**
    - **Secuencia: n,n-1,n-2,n-3… 2,1,0,n,n-1…**
- Al pasar por 0 produce una señal que puede configurarse como interrupción

Registros de SysTick Timer

Address	31-24	23-17	16	15-3	2	1	0	Nombre
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	SYST-CSR - control/stat.
$E000E014	0	24-bit RELOAD value						SYST-RVR – Reload Val.
$E000E018	0	24-bit CURRENT value of SysTick counter						SYST-CVR – Current Val.

- CSR – Registro de Control y Status
  - Enable=0 detiene el reloj. En 1 lo arranca.
  - Se puede elegir varias fuentes de reloj.
  - COUNT es un flag – solo de lectura (escribir no hace nada)
    - se pone a 1 cada vez que el contador pasa por 0.
    - Cuando se lee el contador, se pone a 0.
    - Si pide interrupción, con la respuesta se pone a 0 automáticamente.
  - INTEN=1 para habilitar interrupción cuando COUNT=1
- RELOAD VALUE REGISTER: es el valor de recarga (n)
  - El programa escribe allí el valor máximo que desea para el contador, n
- CURRENT VALUE REGISTER es el valor de la cuenta actual.
  - Escribir cualquier cosa lo pone en 0 → COUNT=0 automáticamente.
  - Al leer COUNT=0

Inicialización: 5 pasos.

- ENABLE=0 para parar el contador.
- Escribir valor de RELOAD (n).
- Resetear Contador y COUNT=0:
  - escribiendo cualquier valor en NVIC_ST_CURRENT_R
- CLK_SRC=1
  - (elige por ej. una fuente de 45 MHz)
- Que cuente ENABLE=1 y habilitar interrupción con INTEN=1 en NVIC_ST_CTRL_R

Ejemplo: INT cada 0,1 seg.

- Si frecuencia clock = 48 MHz.
- T=20,83 ns
- Para esperar 0,1 seg. se necesita
  - 0,1 x 10^9/20,83 = 4.800.000 = n+1
- n = 4.799.999
- Máximo n posible = 2^{24} =16.777.216

Inicializar SysTick Timer

```
systick_init:
    // Inhabilitar SysTick antes de configurar
    LDR R1,=SYST_CSR
    MOV R0,#0
    STR R0,[R1]       // Quitar ENABLE
    // Configurar el desborde para un periodo de 0,1 Segundos
    LDR R1,=SYST_RVR
    LDR R0,=#(4800000-1)
    STR R0,[R1]       // Especificar valor RELOAD
    // Incializar el valor actual del contador
    // Escribir cualquier valor en CVR limpia el contador
    LDR R1,=SYST_CVR
    STR R0,[R1]       // Clear COUNTER y flag COUNT
    // Habilitar el SysTick con el reloj del nucleo
    LDR R1,=SYST_CSR
    MOV R0,#0x05
    STR R0,[R1]       // Fijar ENABLE y CLOCK_SRC, sin INTEN
    BX  LR
```

Espera variable con SysTick

```
/* Rutina de retardo usando espera activa.
 * Input: R0 Espera en ticks del clock interno
 *        48 MHz (20.83 nSec cada tick)
 * Output: Ninguna
 * Modifica: R0,R1,R2 */
systick_wait:
    MOV  R1,#0x00         // INICIALIZAR SYSTICK
    LDR  R2,=SYST_CSR
    STR  R1,[R2]          // Quitar enable
    SUB  R0,R0,#1         // Decremento en 1 el retardo
    STR  R0,[R2,#4]       // Cargo el tiempo de espera (RVR)
    STR  R1,[R2,#8]       // Borro cuenta actual (CVR) y COUNT
    MOV  R1,#0x05
    STR  R1,[R2]          // Activo el contador
systick_loop:
    LDR  R1,[R2]          // es el registro de estado
    ANDS R1,#(1 << 16)    // Verifico COUNT flag (Bit 16)
    BEQ  systick_loop     // Repito hasta que flag es uno
    BX   LR
```

Rutina de espera activa

```
/* Rutina de retardo usando espera activa.
 * Input: R0 Cantidad de décimas de segundo a esperar
 *        48 MHz (20.83 nsec cada tick)
 * Output: Ninguna
 * Modifica: R0,R1,R2,R3 */
delay:
    PUSH {LR}              // Conservo la direccion de retorno
    MOVS R3,R0             // Transfiero la cantidad de ciclos
    BEQ  final         // Si R0=0 retorna
delay_loop:
    LDR  R0,=#4800000      // 0,1 seg a 48 MHz
    BL   systick_wait      // Genero una demora de 100 ms
    SUBS R3,#1             // Decremento el retardo pendiente
    BHI  delay_loop        // Repito mientras sea mayor a cero
final:
    POP  {PC}              // Retono de la subrutina
    .pool
```

Con Interrupciones

- La inicialización es muy parecida, pero ahora se habilitan las interrupciones
- Se recibe una interrupción cada 0,1 ns.
- Para esperar 1 segundo se debe:
  - Fijar el valor de variables en el programa principal
  - ESPERA = 10 y COMPLETO = 0
  - ESPERA es el número de veces que se debe procesar una interrupción para poner COMPLETO=1

Rutina de Interrupción

```
systick_isr:
    LDR  R0,=ESPERA        // Apunta R0 a ESPERA
    LDRB R1,[R0]           // Cargo el valor de ESPERA
    SUBS R1,#1             // Decremento el valor de ESPERA
    BHI  systick_exit      // ESPERA > 0, No pasaron 10 veces
    LDR  R2,=COMPLETO      // Pasaron las 10 veces
    MOV  R1,#1
    STRB R1,[R2]           // Completo = 1
    MOV  R1,#10            // Vuelve a cargar ESPERA con 10
systick_exit:
    STRB R1,[R0]           // Actualiza ESPERA
    BX   LR                // Retorna al programa principal
    .pool
```

¿Por qué no se pasan los parámetros en Registros?

Programa de Espera no activa

```
LDR    R0,=espera      // Apunto R0 a la variable espera
MOV    R1,#10          // Cargo el valor deseado
STRB   R1,[R0]         // Inicializo la variable espera
LDR    R0,=completo    // Apunto R0 a la bandera
MOV    R1,#0
STRB   R1,[R0]         // Inicializo la bandera en false
LDR    R2,=SYST_CSR
LDR    R3,[R2]
    STR    R1,[R2,#8]      // Borro cuenta actual (CVR)
    ORR    R3,#1           // Habilito la cuenta de SysTick
    STR    R3,[R2]         // con ENABLE=1
wait_loop:
    WFI                    // Suspendo la ejecución
    LDRB   R1,[R0]         // Cargo el valor de la bandera
    CMP    R1,#0
    BEQ    wait_loop       // Sigo esperando mientras false
    EOR    R3,#1           // Inhabilito la cuenta
    STR    R3,[R2]         // con ENABLE=0
    …
```

Comentarios Adicionales

- WFI: wait for interrupt, entra en bajo consumo hasta que llega una interrupción.
- En lugar de esperar, el programa puede hacer otras cosas.
- Estas rutinas sirven para recordarle al CPU hacer tareas periódicas, por ejemplo el ingreso de datos.
  - Ej Secretaria con alarma para una ronda de café

Ritual Inicialización I

- El número de excepción es #15.
- **Atención: es interna** → prioridad en otra parte. (manual arquit. ARMv7M)
- Poner prioridad, por ejemplo en 2.
- Hay registros SHPRx
  - Guardan prioridad en un byte.
  - x va de 1 a 3, comienza en Exc. #4. Menores son fijas y no se configuran por Sw.
  - Prioridad < 8 en MCU NXP, se usan 3 MSB.
  - Calcular x = (15-4)/4 + 1 = 3, resto=byte=3
  - Prioridad 010=2 → 01000000 = 0x40
  - Byte 3 de SHPR3 = 0x40

Ritual Inicialización II

```
systick_init:
    CPSID I                 // Deshabilito interrupciones
    // Configurar prioridad de la interrupción
    LDR R1,=SHPR3           // Apunto al registro de prioridades
    LDR R0,[R1]             // Cargo las prioridades actuales
    MOV R2,#2               // Fijo la prioridad en 2
    BFI R0,R2,#29,#3        // Inserto el valor en el campo
    STR R0,[R1]             // Actualizo las prioridades
    ... // Habilitar el SysTick con el reloj del núcleo
    ... // que INTEN=1, habrá interrupciones.
    ... // y que no cuente (ENABLE=0)
    ... // Configurar el desborde para el periodo deseado
    ... // Inicializar el valor actual del contador
    CPSIE I                 // Rehabilito interrupciones
    BX    LR                // Retorno al programa principal
```

Ej Reloj Interno.

- Rutina ingreso_hora pone en la variable global “hora”: 3 bytes con hh,mm,ss respectivamente.
- Mas adelante veremos ingreso_hora.
- **Mantenerlo actualizado con el tiempo**.
- Estrategia.
  - Escribir rutina “actualiza_reloj”
  - Que sea convocada desde systick_isr cada seg
  - Ya no se utilizará el flag de COMPLETO
  - Ejemplo de una tarea periódica.
- En este ej. Inicializar SisTick para que cuente

Rutina de interrupción

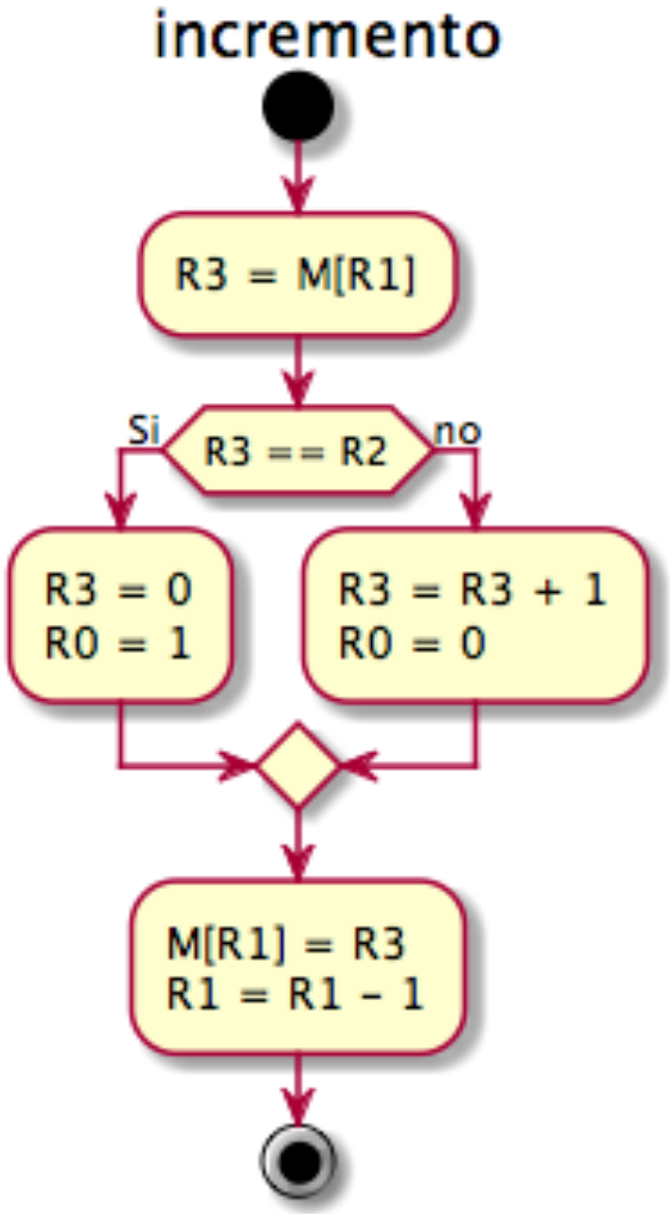
```
systick_isr:
    LDR  R0,=espera          // Apunta R0 a espera
    LDRB R1,[R0]             // Carga el valor de espera
    SUBS R1,#1               // Decrementa el valor de espera
    BHI  systick_exit        // Espera > 0, No pasaron 10 veces
    PUSH {R0,LR}             // Conserva los registros que uso
    BL   actualiza_reloj     // Llama a la subrutina de fecha
    POP  {R0,LR}             // Recupera los registros
    MOV  R1,#10              // carga espera con 10 iteraciones
systick_exit:
    STRB R1,[R0]             // Actualiza la variable espera
    BX   LR                  // Retorna al programa principal
```

Algoritmo actualización reloj

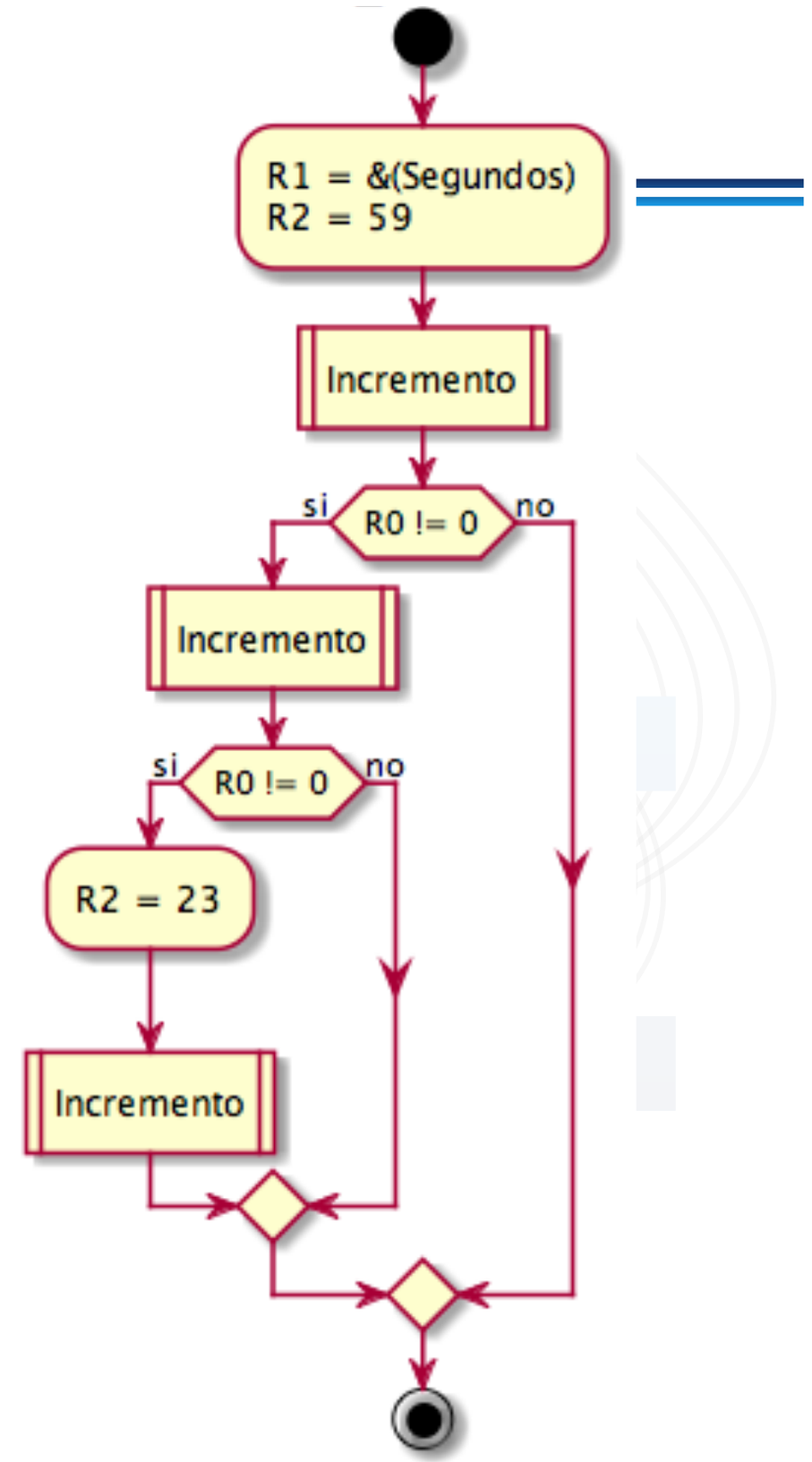
1) If ss<59 then ss=ss+1, fin
2) ss=0
3) If mm<59 then mm=mm+1, fin
4) mm=0
5) If hh<23 then hh=hh+1, fin
6) hh=0
7) Fin

- 1,2 – 3,4 – 5,6 se parecen: subrutina.

Diagramas de flujo



actualiza_reloj



Rutina para actualizar reloj

```
actualiza_reloj:
    PUSH {LR}              // Conservar la dirección de retorno
    LDR  R1,=hora+2        // Apuntar R1 a los segundos
    MOV  R2,#59            // Cargar el valor máximo de segundos
    BL   incremento        // Incrementar segundo con limite
    CBZ  R0,final          // Si no hubo desborde terminar
    BL   incremento        // Incrementar minutos con limite
    CBZ  R0,final          // Si no hubo desborde terminar
    MOV  R2,#23            // Carga el valor máximo de las horas
    BL   incremento        // Incrementar con limite las horas
final:
    POP  {PC}              // Retornar recuperando PC de la pila
```

Rutina de incremento

```
incremento:
    LDRB   R3,[R1]          // Cargar el numero a incrementar
    CMP    R3,R2            // Comparar el numero con el limite
    ITTEE  EQ               // Si el numero es igual al limite
    MOVEQ  R3,#0            // Entonces el numero vale cero
    MOVEQ  R0,#1            // Entonces resultado es true
    ADDNE  R3,#1            // Sino incrementar el numero
    MOVNE  R0,#0            // Sino resultado es false
    STRB   R3,[R1],#-1      // Guardar y actualizar el puntero
    BX LR
```

Latencia de la Interrupción

- Es el tiempo que demora el CPU desde que llega el pedido hasta que lo atiende, suponiendo que el pedido no quedó pendiente.
- En Cortex-M4, en general la latencia será de12 T, puede llegar a 23T si usa la unidad de punto flotante.
- En el peor caso se debe incluir ademas la instrucción más larga, un branch de 3T.
- Si hay anidamiento también será de12T, pero puede ser menor en interrupciones simultáneas.
- Si el timer es continuo, la latencia no es acumulable y el retraso va a ser siempre el mismo.

Reset (repaso)

- ¿Qué pasa cuando se da tensión al μC?
- Problemas a Solucionar:
  - Registros y RAM indeterminados.
  - ¡Por tanto PC indeterminado!
  - Interfases E/S indeterminados.
- Es necesario asegurar un valor inicial al PC, SP, puertos y dispositivos μC.
- Que PC apunte a una Rutina de inicialización.

RESET en Cortex-M4

- Hay un Reg en NVIC para pedir reset por Sw.
- Existe una entrada al MCU para RESET x Hw
- Se activa por un cierto tiempo (POR circuit)
  - Para que se estabilice el clock interno.
- A continuación:
  - Interrupciones inhabilitadas ¿por qué?
  - Modo supervisor ¿por qué?
  - BASEPRI=0
  - MSP←M(0)
  - PC←M(4). PC apunta a rutina de booteo.
  - Tabla de Vectores y Rutina Booteo en FLASH

Un poco de historia

- El concepto que nosotros denominamos TRAP se introduce en 1951 con la Univac 1103A
- El concepto de interrupciones se introduce en 1954 en la NBS DYSEAC, una computadora portátil que se podía transportar en dos trailers: uno de 12 y otro 8 toneladas.
- En 1955 se presenta el uso de traps para permitir la depuración de un programa.
- En 1957 Dijkstra propone el sistema de interrupciones vectorizadas y un enfoque unificado con los saltos condicionales.