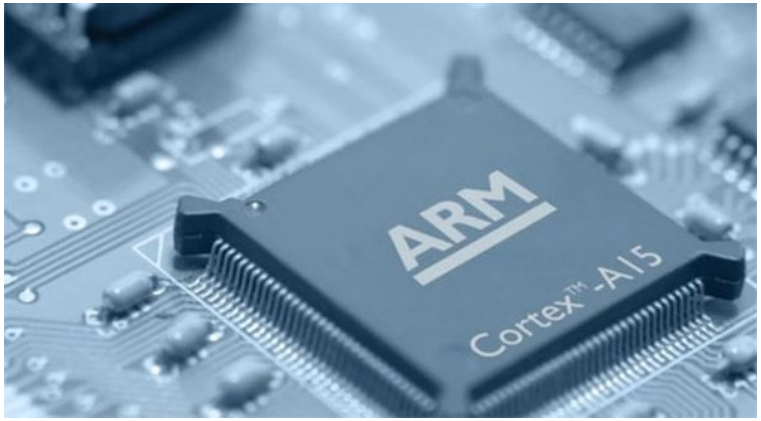


Procesadores ARM - System Programming

Alejandro Furfaro

15 de junio de 2020

Temario

Temario

Desarrollo de un Sistema Embebido

Desarrollo de un Sistema Embebido

- Diseño de Hardware

Desarrollo de un Sistema Embebido

- Diseño de Hardware
- Diseño de Firmware

Desarrollo de un Sistema Embebido

- Diseño de Hardware
- Diseño de Firmware
- Diseño de un Sistema Operativo

Desarrollo de un Sistema Embebido

- Diseño de Hardware
- Diseño de Firmware
- Diseño de un Sistema Operativo
- Diseño de las tareas que resuelven la funcionalidad

Desarrollo de un Sistema Embebido

- Diseño de Hardware
- Diseño de Firmware
- Diseño de un Sistema Operativo
- Diseño de las tareas que resuelven la funcionalidad

De donde se parte

El embrión del desarrollo de un sistema es el diseño del Hardware y del Software de mas bajo nivel que va a ejecutarse ni bien se enciende el dispositivo. A este último se lo denomina Firmware. Y es la primer etapa de desarrollo de software que da vida al sistema.

Firmware

Firmware

- Reside en memoria No Volátil, toma el control en el encendido o luego de un Reset y realiza la inicialización de bajo nivel.

Firmware

- Reside en memoria No Volátil, toma el control en el encendido o luego de un Reset y realiza la inicialización de bajo nivel.
- Luego cede el control, y permanece disponible proveyendo a demanda servicios de acceso al hardware.

Firmware

- Reside en memoria No Volátil, toma el control en el encendido o luego de un Reset y realiza la inicialización de bajo nivel.
- Luego cede el control, y permanece disponible proveyendo a demanda servicios de acceso al hardware.
- Dependiendo de la especificación del sistema, el firmware cederá el control a un modesto microkernel que maneje un grupo fijo de tareas (en ocasiones sin distinguir modo user de modo privilegiado),o a algún Sistema Operativo mas sofisticado (Ej: Linux).

Firmware

- Reside en memoria No Volátil, toma el control en el encendido o luego de un Reset y realiza la inicialización de bajo nivel.
- Luego cede el control, y permanece disponible proveyendo a demanda servicios de acceso al hardware.
- Dependiendo de la especificación del sistema, el firmware cederá el control a un modesto microkernel que maneje un grupo fijo de tareas (en ocasiones sin distinguir modo user de modo privilegiado),o a algún Sistema Operativo mas sofisticado (Ej: Linux).
- En este último caso no confundir Firmware con Boot Loader. Este último es solo la parte que carga el sistema operativo y le cede el control. Ejecuta en RAM y se remueve una vez cargado el sistema operativo (Por mas o menos sofisticado que éste último resulte).

Etapas del Firmware

Etapas del Firmware

- Inicializar la plataforma de Hardware

Etapas del Firmware

- Inicializar la plataforma de Hardware
- Establecer una capa de abstracción del hardware

Etapas del Firmware

- Inicializar la plataforma de Hardware
- Establecer una capa de abstracción del hardware
- Cargar la imagen del sistema

Etapas del Firmware

- Inicializar la plataforma de Hardware
- Establecer una capa de abstracción del hardware
- Cargar la imagen del sistema
- Ceder el control

Inicializar la plataforma de Hardware

Inicializar la plataforma de Hardware

- Comprobar que la plataforma ha sido correctamente inicializada. Esto es comprobar que los registros de control de un procesador particular estén mapeados en la dirección de memoria apropiada, y remapear la memoria a un layout esperado si se requiere.

Inicializar la plataforma de Hardware

- Comprobar que la plataforma ha sido correctamente inicializada. Esto es comprobar que los registros de control de un procesador particular estén mapeados en la dirección de memoria apropiada, y remapear la memoria a un layout esperado si se requiere.

- Identificar el core de la plataforma leyendo el registro 0 del Coprocesador 15 (allí está el tipo exacto del core y su fabricante).

Inicializar la plataforma de Hardware

- Comprobar que la plataforma ha sido correctamente inicializada. Esto es comprobar que los registros de control de un procesador particular estén mapeados en la dirección de memoria apropiada, y remapear la memoria a un layout esperado si se requiere.

- Identificar el core de la plataforma leyendo el registro 0 del Coprocesador 15 (allí está el tipo exacto del core y su fabricante).

- Identificar la plataforma. (ya sea leyendo un par de periféricos característicos o accediendo al chip y buscando allí su ID)

Inicializar la plataforma de Hardware

- Comprobar que la plataforma ha sido correctamente inicializada. Esto es comprobar que los registros de control de un procesador particular estén mapeados en la dirección de memoria apropiada, y remapear la memoria a un layout esperado si se requiere.
- Identificar el core de la plataforma leyendo el registro 0 del Coprocesador 15 (allí está el tipo exacto del core y su fabricante).
- Identificar la plataforma. (ya sea leyendo un par de periféricos característicos o accediendo al chip y buscando allí su ID)
- Ejecutar un código de diagnóstico (asegurar que todos los recursos funcionan apropiadamente (esto es 100 % plataforma dependiente)

Inicializar la plataforma de Hardware

- Opcionalmente puede proveerse alguna facilidad de debuging

Inicializar la plataforma de Hardware

- Opcionalmente puede proveerse alguna facilidad de debuging
- En tal caso se requiere una interfaz de tipo interactiva (**CLI C**ommand **L**ine **I**ntepreter), que reciba comandos y pueda mostrar los registros del core, la memoria, (escribir o leer), Desensamblar código etc.

Inicializar la plataforma de Hardware

- Opcionalmente puede proveerse alguna facilidad de debuging
- En tal caso se requiere una interfaz de tipo interactiva (**CLI C**ommand **L**ine **I**ntepreter), que reciba comandos y pueda mostrar los registros del core, la memoria, (escribir o leer), Desensamblar código etc.
- Es deseable un firmware sofisticado que pueda proveer una interfaz **CLI** para el Sistema Operativo.

Hardware Abstraction Layer

HAL

Proporciona una **API** (**A**pplication **P**rogramming **I**nterface), para que tanto el Sistema Operativo (independientemente de su escala), y eventualmente las tareas (si no hay restricciones por parte del Sistema Operativo para acceder al hardware), puedan ganar acceso a los diferentes recursos de Hardware sin necesidad de conocer sus detalles de configuración (direcciones de los registros de cada periférico, organización de los campos de bits de sus registros de estados y comandos, etc).

Hardware Abstraction Layer

HAL

Proporciona una **API** (**A**pplication **P**rogramming **I**nterface), para que tanto el Sistema Operativo (independientemente de su escala), y eventualmente las tareas (si no hay restricciones por parte del Sistema Operativo para acceder al hardware), puedan ganar acceso a los diferentes recursos de Hardware sin necesidad de conocer sus detalles de configuración (direcciones de los registros de cada periférico, organización de los campos de bits de sus registros de estados y comandos, etc).

- El API debe estar definida de manera que se mantenga igual independientemente del hardware de base y del sistema operativo que se empleen en cada sistema.

Hardware Abstraction Layer

HAL

Proporciona una **API** (**A**pplication **P**rogramming **I**nterface), para que tanto el Sistema Operativo (independientemente de su escala), y eventualmente las tareas (si no hay restricciones por parte del Sistema Operativo para acceder al hardware), puedan ganar acceso a los diferentes recursos de Hardware sin necesidad de conocer sus detalles de configuración (direcciones de los registros de cada periférico, organización de los campos de bits de sus registros de estados y comandos, etc).

- El API debe estar definida de manera que se mantenga igual independientemente del hardware de base y del sistema operativo que se empleen en cada sistema.
- Caracter modular. Un bloque de software por cada periférico: Device Driver.

Cargar la imagen del sistema

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.
- En general a mayor simpleza, ejecutan en la misma Memoria No Volátil en la que se los graba. Tal el caso de la familia Cortex-M

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.
- En general a mayor simpleza, ejecutan en la misma Memoria No Volátil en la que se los graba. Tal el caso de la familia Cortex-M
- En general todos traen una Flash ROM y allí mediante una prestación común conocida como **F**lash ROM **F**illing **S**ystem (**FFS**), se puede mantener varias imágenes diferentes en la misma memoria.

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.
- En general a mayor simpleza, ejecutan en la misma Memoria No Volátil en la que se los graba. Tal el caso de la familia Cortex-M
- En general todos traen una Flash ROM y allí mediante una prestación común conocida como **F**lash ROM **F**illing **S**ystem (**FFS**), se puede mantener varias imágenes diferentes en la misma memoria.
- Los sistemas de mas alta gama suelen traer un lector de SD. El controlador permite ver a esa SD como si fuese un disco.

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.
- En general a mayor simpleza, ejecutan en la misma Memoria No Volátil en la que se los graba. Tal el caso de la familia Cortex-M
- En general todos traen una Flash ROM y allí mediante una prestación común conocida como **F**lash ROM **F**illing **S**ystem (**FFS**), se puede mantener varias imágenes diferentes en la misma memoria.
- Los sistemas de mas alta gama suelen traer un lector de SD. El controlador permite ver a esa SD como si fuese un disco.
- En tal caso, el firmware debe incluir un driver de disco, y ser capaz de comprender el formato del File System de la SD.

Cargar la imagen del sistema

- Dependen del tipo de medio en el que reside la imagen.
- No todos los Sistemas Operativo necesitan copiarse a RAM.
- En general a mayor simpleza, ejecutan en la misma Memoria No Volátil en la que se los graba. Tal el caso de la familia Cortex-M
- En general todos traen una Flash ROM y allí mediante una prestación común conocida como **F**lash ROM **F**illing **S**ystem (**FFS**), se puede mantener varias imágenes diferentes en la misma memoria.
- Los sistemas de mas alta gama suelen traer un lector de SD. El controlador permite ver a esa SD como si fuese un disco.
- En tal caso, el firmware debe incluir un driver de disco, y ser capaz de comprender el formato del File System de la SD.
- El File System también puede estar en un server remoto accesible vía red de datos. En este caso el firmware deberá además disponer de un driver de Ethernet.

Cargar la imagen del sistema

- El firmware debe considerar una vez ubicado el archivo de la imagen, el formato de ésta.

Cargar la imagen del sistema

- El firmware debe considerar una vez ubicado el archivo de la imagen, el formato de ésta.

- Lo mas simple y directo es proveer una imagen binaria plana, es decir, sin ningun tipo de encabezado. La imagen exacta de como va en memoria todo el sistema a cargar.

Cargar la imagen del sistema

- El firmware debe considerar una vez ubicado el archivo de la imagen, el formato de ésta.
- Lo mas simple y directo es proveer una imagen binaria plana, es decir, sin ningun tipo de encabezado. La imagen exacta de como va en memoria todo el sistema a cargar.
- Si nuestro sistema va a cargar Linux o cualquier sistema UNIX like deberá entonces lidiar con imágenes con formato. En este caso el formato es **ELF** (**E**xecutable and **L**inkable **F**ormat).

Cargar la imagen del sistema

- El firmware debe considerar una vez ubicado el archivo de la imagen, el formato de ésta.
- Lo mas simple y directo es proveer una imagen binaria plana, es decir, sin ningun tipo de encabezado. La imagen exacta de como va en memoria todo el sistema a cargar.
- Si nuestro sistema va a cargar Linux o cualquier sistema UNIX like deberá entonces lidiar con imágenes con formato. En este caso el formato es **ELF** (**E**xecutable and **L**inkable **F**ormat).
- Finalmente la imagen puede estar comprimida y/o encriptada. Mas complejidad eventualmente.

Ceder el control

Ceder el control

- El firmware finalmente una vez cargada la imagen en RAM debe invocar al punto de entrada de la misma para transferirle el control.

Ceder el control

- El firmware finalmente una vez cargada la imagen en RAM debe invocar al punto de entrada de la misma para transferirle el control.
- Una vez hecho esto, el software de inicialización y carga de la imagen se inactiva.

Ceder el control

- El firmware finalmente una vez cargada la imagen en RAM debe invocar al punto de entrada de la misma para transferirle el control.
- Una vez hecho esto, el software de inicialización y carga de la imagen se inactiva.
- Sin embargo el **HAL** permanecerá activo y disponible como medio de acceso al hardware, mediante la instrucción **`SVC`** (o **`SWI`** en sistemas mas antiguos).

Ceder el control

- El firmware finalmente una vez cargada la imagen en RAM debe invocar al punto de entrada de la misma para transferirle el control.
- Una vez hecho esto, el software de inicialización y carga de la imagen se inactiva.
- Sin embargo el **HAL** permanecerá activo y disponible como medio de acceso al hardware, mediante la instrucción **`SVC`** (o **`SWI`** en sistemas mas antiguos).
- Transferir el control implica actualizar el vector de interrupciones con los handlers apropiados del Sistema Operativo, y modificar el valor del Registro **`PC`**

Ceder el control

- El firmware finalmente una vez cargada la imagen en RAM debe invocar al punto de entrada de la misma para transferirle el control.
- Una vez hecho esto, el software de inicialización y carga de la imagen se inactiva.
- Sin embargo el **HAL** permanecerá activo y disponible como medio de acceso al hardware, mediante la instrucción **`SVC`** (o **`SWI`** en sistemas mas antiguos).
- Transferir el control implica actualizar el vector de interrupciones con los handlers apropiados del Sistema Operativo, y modificar el valor del Registro **`PC`**
- Si el sistema operativo que se va a bootear es LINUX, entonces además hay que pasarle al kernel una estructura de datos binaria llamada *Device Tree* que contiene la descripción completa del hardware del sistema (CPU, Memoria y cada Device de E/S)

Temario

Que tenemos a mano para hacer esta tarea

Que tenemos a mano para hacer esta tarea

- Para escribir un firmware como el descripto solo contamos con nuestro conocimiento de Arquitectura y Organización de Computadores (**AyOC**). Insumo fundamental.

Que tenemos a mano para hacer esta tarea

- Para escribir un firmware como el descripto solo contamos con nuestro conocimiento de Arquitectura y Organización de Computadores (**AyOC**). Insumo fundamental.
- Tener presente que el conocimiento es como el dinero. Nunca es suficiente.

Que tenemos a mano para hacer esta tarea

- Para escribir un firmware como el descripto solo contamos con nuestro conocimiento de Arquitectura y Organización de Computadores (**AyOC**). Insumo fundamental.
- Tener presente que el conocimiento es como el dinero. Nunca es suficiente.
- Seguir profundizando tus conocimientos de manera contínua. Cada día.

Que tenemos a mano para hacer esta tarea

- Para escribir un firmware como el descripto solo contamos con nuestro conocimiento de Arquitectura y Organización de Computadores (**AyOC**). Insumo fundamental.
- Tener presente que el conocimiento es como el dinero. Nunca es suficiente.
- Seguir profundizando tus conocimientos de manera contínua. Cada día.
- Además de ello necesitamos la documentación del hardware de base en particular sobre el que vamso a trabajar, un compilador, un linker y alguna otra herramientas de desarrollo.

Que tenemos a mano para hacer esta tarea

- Para escribir un firmware como el descripto solo contamos con nuestro conocimiento de Arquitectura y Organización de Computadores (**AyOC**). Insumo fundamental.
- Tener presente que el conocimiento es como el dinero. Nunca es suficiente.
- Seguir profundizando tus conocimientos de manera contínua. Cada día.
- Además de ello necesitamos la documentación del hardware de base en particular sobre el que vamso a trabajar, un compilador, un linker y alguna otra herramientas de desarrollo.
- Eso. Nada mas (y nada menos).

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.
- **Respuesta**: Si, claro. ¿Cual es el punto?

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.
- **Respuesta**: Si, claro. ¿Cual es el punto?
- **Pregunta**: Nada.... solo que... ¿Para que necesitamos entonces meternos en este problema?

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.
- **Respuesta**: Si, claro. ¿Cual es el punto?
- **Pregunta**: Nada.... solo que... ¿Para que necesitamos entonces meternos en este problema?
- **Respuesta**: Para aprender.

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.
- **Respuesta**: Si, claro. ¿Cual es el punto?
- **Pregunta**: Nada.... solo que... ¿Para que necesitamos entonces meternos en este problema?
- **Respuesta**: Para aprender.
- **Pregunta**: ¿No hay una inicialización que provee el fabricante? ¿Para que nos vamos a poner a hacer la nuestra?.

Preguntas que es esperable te estés haciendo

- **Pregunta**: ¿Pero esto no venía ya desarrollado?.
- **Respuesta**: Si, claro. ¿Cual es el punto?
- **Pregunta**: Nada.... solo que... ¿Para que necesitamos entonces meternos en este problema?
- **Respuesta**: Para aprender.
- **Pregunta**: ¿No hay una inicialización que provee el fabricante? ¿Para que nos vamos a poner a hacer la nuestra?.
- **Respuesta**: Para aprender.

Bienvenido a la Ingeniería

- Ahora pregunto yo:

Bienvenido a la Ingeniería

- Ahora pregunto yo:
- ¿Como suponés que se genera el código de inicialización de un sistema de los que utilizas a diario? ¿Solo?

Bienvenido a la Ingeniería

- Ahora pregunto yo:
- ¿Como suponés que se genera el código de inicialización de un sistema de los que utilizas a diario? ¿Solo?
- Porque si algo es cierto, es que el desarrollo de cualquier computador comienza desde cero con ese tipo de código.

Bienvenido a la Ingeniería

- Ahora pregunto yo:
- ¿Como suponés que se genera el código de inicialización de un sistema de los que utilizas a diario? ¿Solo?
- Porque si algo es cierto, es que el desarrollo de cualquier computador comienza desde cero con ese tipo de código.
- Esto aplica a tu PC, a la placa de Info II, o a cualquier otro computador, kit o placa de desarrollo con que hayas interactuado en tu vida.

Bienvenido a la Ingeniería

- Ahora pregunto yo:
- ¿Como suponés que se genera el código de inicialización de un sistema de los que utilizas a diario? ¿Solo?
- Porque si algo es cierto, es que el desarrollo de cualquier computador comienza desde cero con ese tipo de código.
- Esto aplica a tu PC, a la placa de Info II, o a cualquier otro computador, kit o placa de desarrollo con que hayas interactuado en tu vida.
- ¿De verdad nunca te detuviste a pensar quien escribe ese código de inicialización?

Bienvenido a la Ingeniería

- Ahora pregunto yo:
- ¿Como suponés que se genera el código de inicialización de un sistema de los que utilizas a diario? ¿Solo?
- Porque si algo es cierto, es que el desarrollo de cualquier computador comienza desde cero con ese tipo de código.
- Esto aplica a tu PC, a la placa de Info II, o a cualquier otro computador, kit o placa de desarrollo con que hayas interactuado en tu vida.
- ¿De verdad nunca te detuviste a pensar quien escribe ese código de inicialización?

A ver...
¿Que profesión tendrá esa persona? Pensá...

Estudios del que escribe código de inicialización.

Indicar su respuesta. (Solo una es válida)

Estudios del que escribe código de inicialización.

Indicar su respuesta. (Solo una es válida)

- **a.** Bombero
- **b.** Odontólogo
- **c.** Fisioterapeuta
- **d.** Chef
- **e.** Ingeniero en Electrónica
- **f.** Vendedor de Seguros
- **g.** Periodista
- **h.** Abogado
- **i.** Actor

Es hora de dejar de pensar y actuar como usuarios.

Diagnóstico

El criterio “usar las bibliotecas del fabricante”, permite cumplir restricciones severas de Time To Market. Esto es muy conveniente en esos casos. Nos evita dificultades.
El costo: El sistema es una caja negra. Y peor aún, como esta comodidad es tentadora, la terminamos adoptando como regla general aun cuando no hay restricciones de tiempo, . . . y terminamos pensando como usuarios finales.
El confort es el principal inhibidor del aprendizaje.

En nuestra PC siempre actuamos como usuarios.

En nuestra PC siempre actuamos como usuarios.

- ¿O acaso alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Están los Wizards!!.

En nuestra PC siempre actuamos como usuarios.

- ¿O acaso alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Están los Wizards!!.
- Buscamos el ícono llamado *Install*, *Setup*, o algo similar, al encontrarlo Doble click para ejecutar el instalador, y click a morir al Botón Aceptar.

En nuestra PC siempre actuamos como usuarios.

- ¿O acaso alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Están los Wizards!!.
- Buscamos el ícono llamado *Install*, *Setup*, o algo similar, al encontrarlo Doble click para ejecutar el instalador, y click a morir al Botón Aceptar.
- ¿Problemas para ejecutar?... desinstalar y volver a instalar,... o botón de reset. 🥲

En nuestra PC siempre actuamos como usuarios.

- ¿O acaso alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Están los Wizards!!.
- Buscamos el ícono llamado *Install*, *Setup*, o algo similar, al encontrarlo Doble click para ejecutar el instalador, y click a morir al Botón Aceptar.
- ¿Problemas para ejecutar?... desinstalar y volver a instalar,... o botón de reset. 🥲
- Las aplicaciones tan digeridas nos llevan a tomar un computador como una caja negra.

En nuestra PC siempre actuamos como usuarios.

- ¿O acaso alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Están los Wizards!!.
- Buscamos el ícono llamado *Install*, *Setup*, o algo similar, al encontrarlo Doble click para ejecutar el instalador, y click a morir al Botón Aceptar.
- ¿Problemas para ejecutar?... desinstalar y volver a instalar,... o botón de reset. 🥲
- Las aplicaciones tan digeridas nos llevan a tomar un computador como una caja negra.
- No estoy diciendo que está mal. Solo expongo lo hechos.

Cuando programamos actuamos como usuarios.

Cuando programamos actuamos como usuarios.

- Pedimos recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.

Cuando programamos actuamos como usuarios.

- Pedimos recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
- Enviamos requerimientos para acceder a la E/S. Esto también.

Cuando programamos actuamos como usuarios.

- Pedimos recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
- Enviamos requerimientos para acceder a la E/S. Esto también.
- Usamos bibliotecas de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.

Cuando programamos actuamos como usuarios.

- Pedimos recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
- Enviamos requerimientos para acceder a la E/S. Esto también.
- Usamos bibliotecas de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.
- Cuando programamos cotidianamente nuestro comportamiento es de usuario final. Siempre recurrimos a código desarrollado “por un fabricante”.

Cuando programamos actuamos como usuarios.

- Pedimos recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
- Enviamos requerimientos para acceder a la E/S. Esto también.
- Usamos bibliotecas de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.
- Cuando programamos cotidianamente nuestro comportamiento es de usuario final. Siempre recurrimos a código desarrollado “por un fabricante”.
- **Y no está mal!!**. Solo advierto que en ocasiones hay que romper las costumbres.

Rompiendo el cascarón

Resultado

Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

Rompiendo el cascarón

Resultado

Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

- Para usuarios generales el comportamiento descripto en los dos o tres slides anteriores es el esperable.

Rompiendo el cascarón

Resultado

Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

- Para usuarios generales el comportamiento descripto en los dos o tres slides anteriores es el esperable.
- Uds. son **otro público**, básicamente porque **eligieron serlo**. ¿Porque están aquí sino?

Rompiendo el cascarón

Resultado

Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

- Para usuarios generales el comportamiento descripto en los dos o tres slides anteriores es el esperable.
- Uds. son **otro público**, básicamente porque **eligieron serlo**. ¿Porque están aquí sino?
- Nuestro trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos, para poderlos diseñar, mejorar, o corregir.

Rompiendo el cascarón

Resultado

Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

- Para usuarios generales el comportamiento descripto en los dos o tres slides anteriores es el esperable.
- Uds. son **otro público**, básicamente porque **eligieron serlo**. ¿Porque están aquí sino?
- Nuestro trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos, para poderlos diseñar, mejorar, o corregir.
- En este caso es un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.

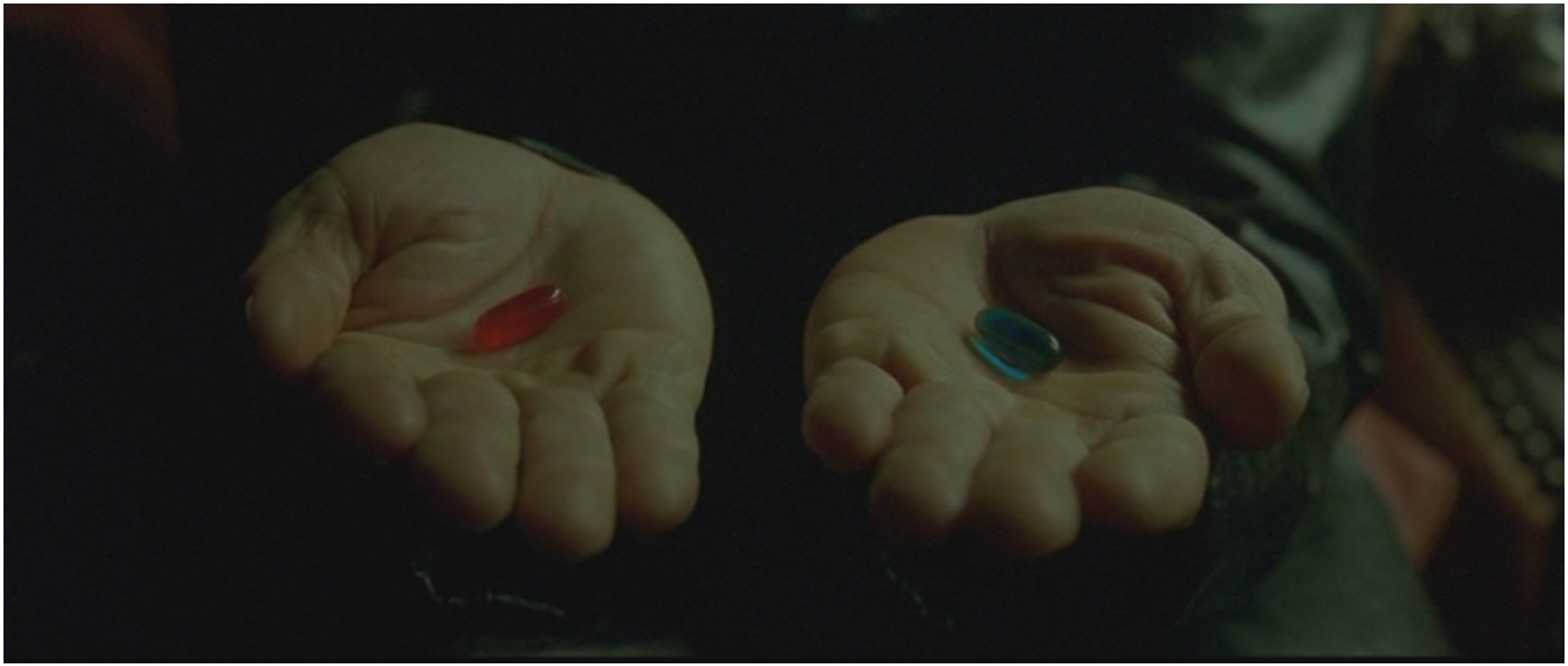
Rompiendo el cascarón

Resultado

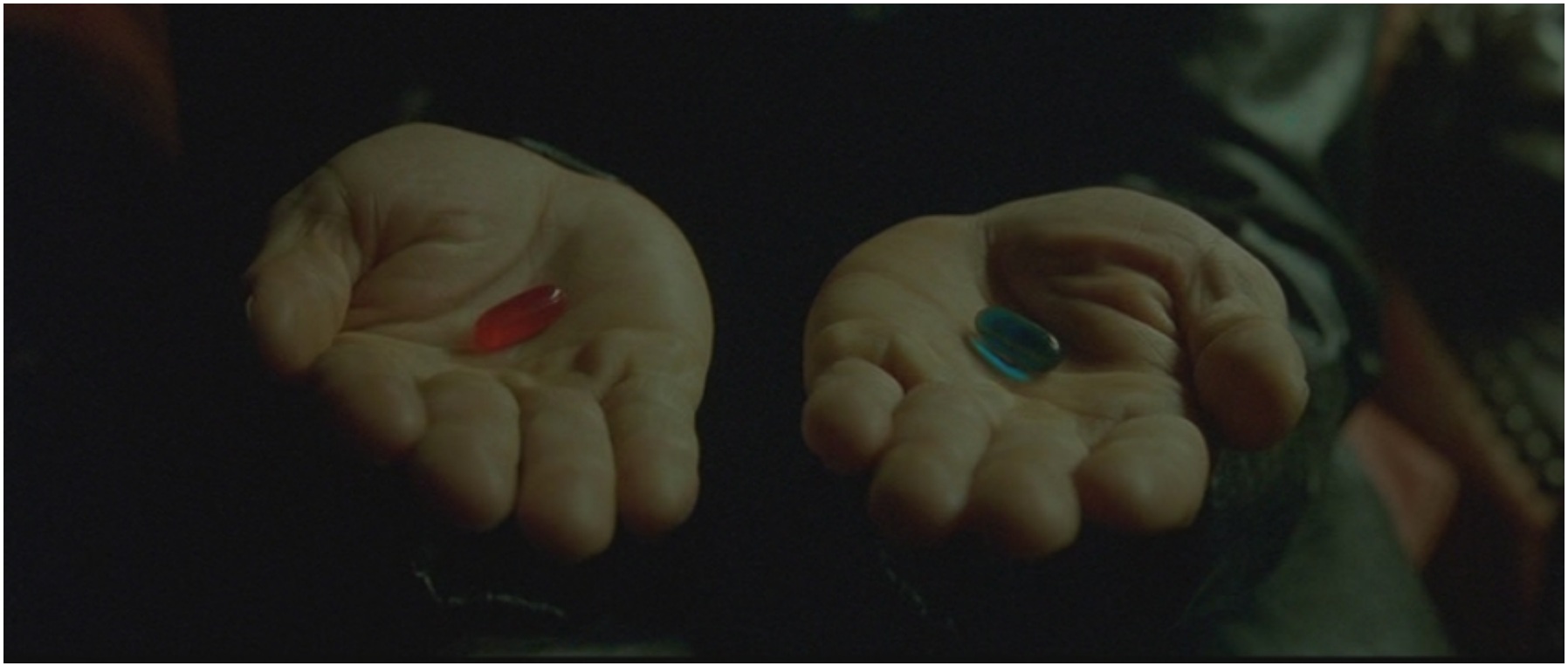
Todo nos conduce a pensar en términos simplistas y tender a trabajar en el nivel mas abstracto posible. Nos desentendemos del harware, y del software que lo acciona de manera directa. ¿Justamente nosotros?

- Para usuarios generales el comportamiento descripto en los dos o tres slides anteriores es el esperable.
- Uds. son **otro público**, básicamente porque **eligieron serlo**. ¿Porque están aquí sino?
- Nuestro trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos, para poderlos diseñar, mejorar, o corregir.
- En este caso es un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar una tecnología. Algo que pesando como usuario nunca vamos a conseguir...

Es como aquella inolvidable escena



Es como aquella inolvidable escena

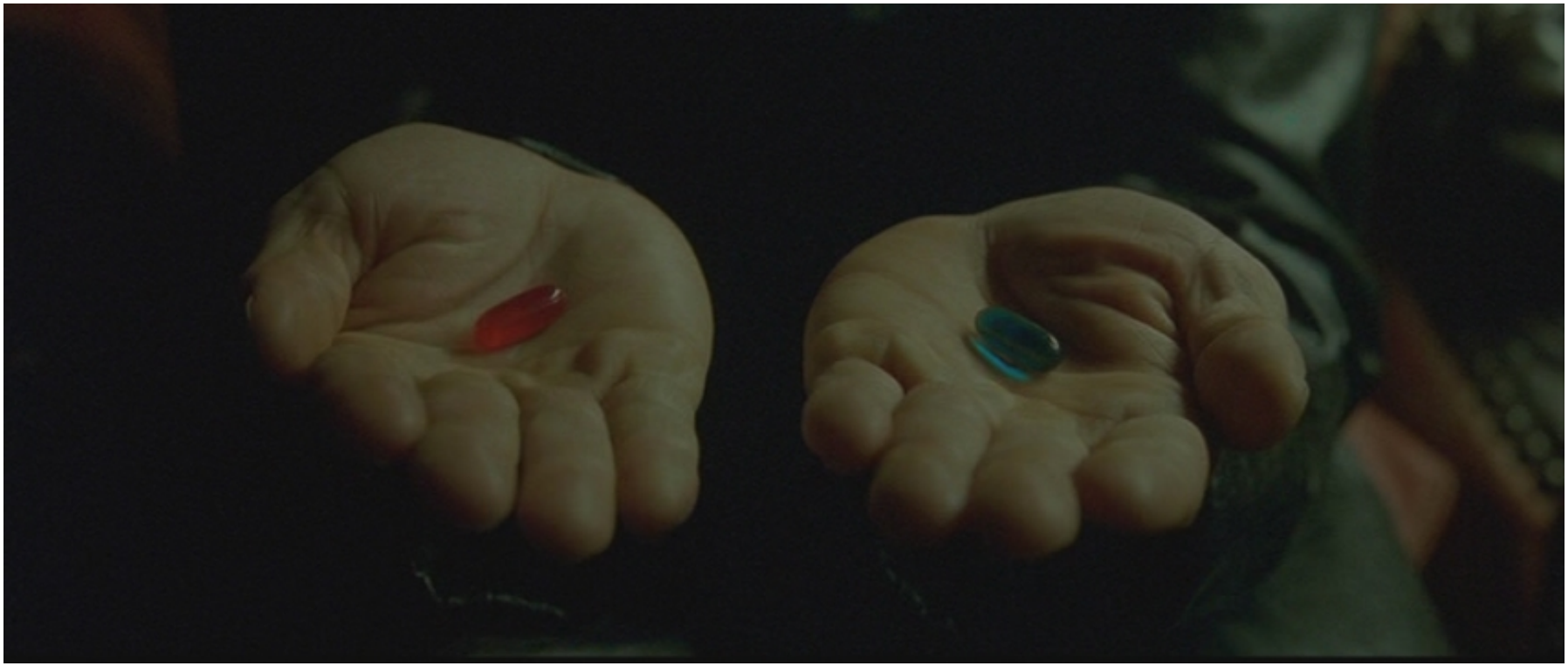


The choice

O nos quedamos con “el código del fabricante”que nos resuelve la vida sin tener que pensar. . .

.

Es como aquella inolvidable escena



The choice

O nos quedamos con “el código del fabricante”que nos resuelve la vida sin tener que pensar. . .
O enfrentamos las cosas como son realmente, las entendemos, y aprendemos, si es necesario haciendo todo desde cero y a pulmón.

¿Como encarar la tarea?

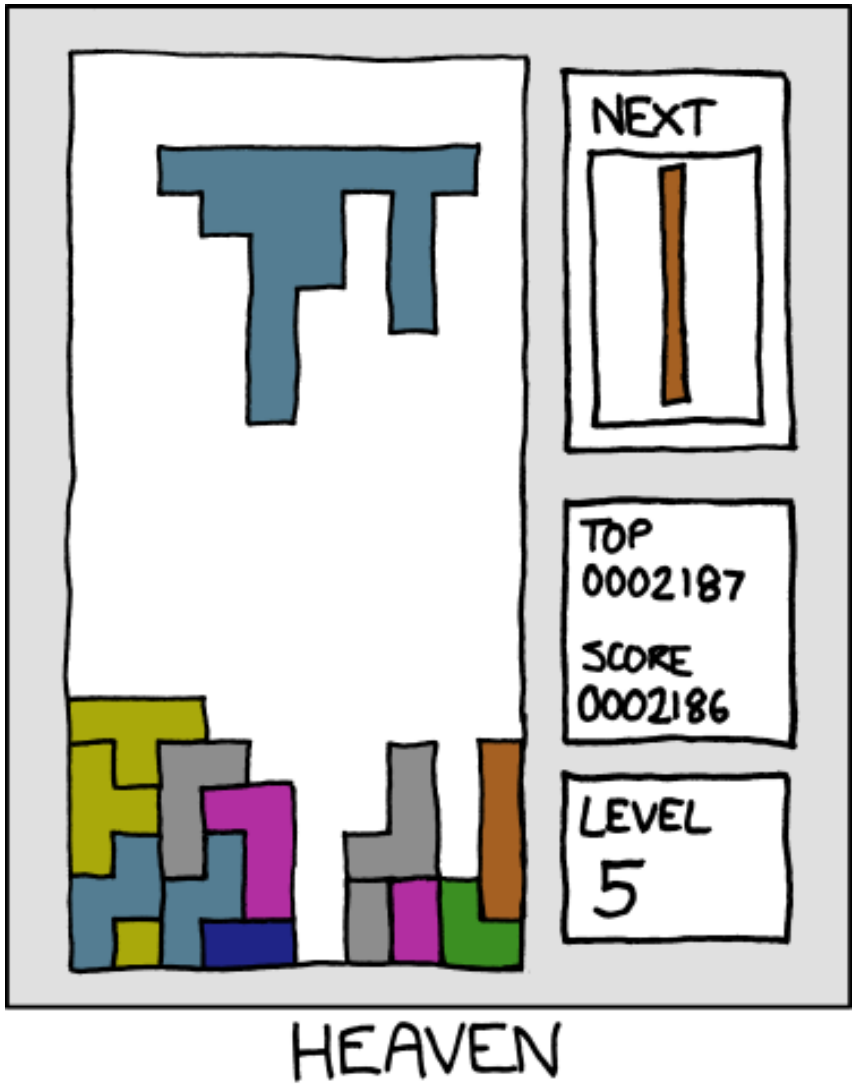
¿Como encarar la tarea?

Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.

¿Como encarar la tarea?

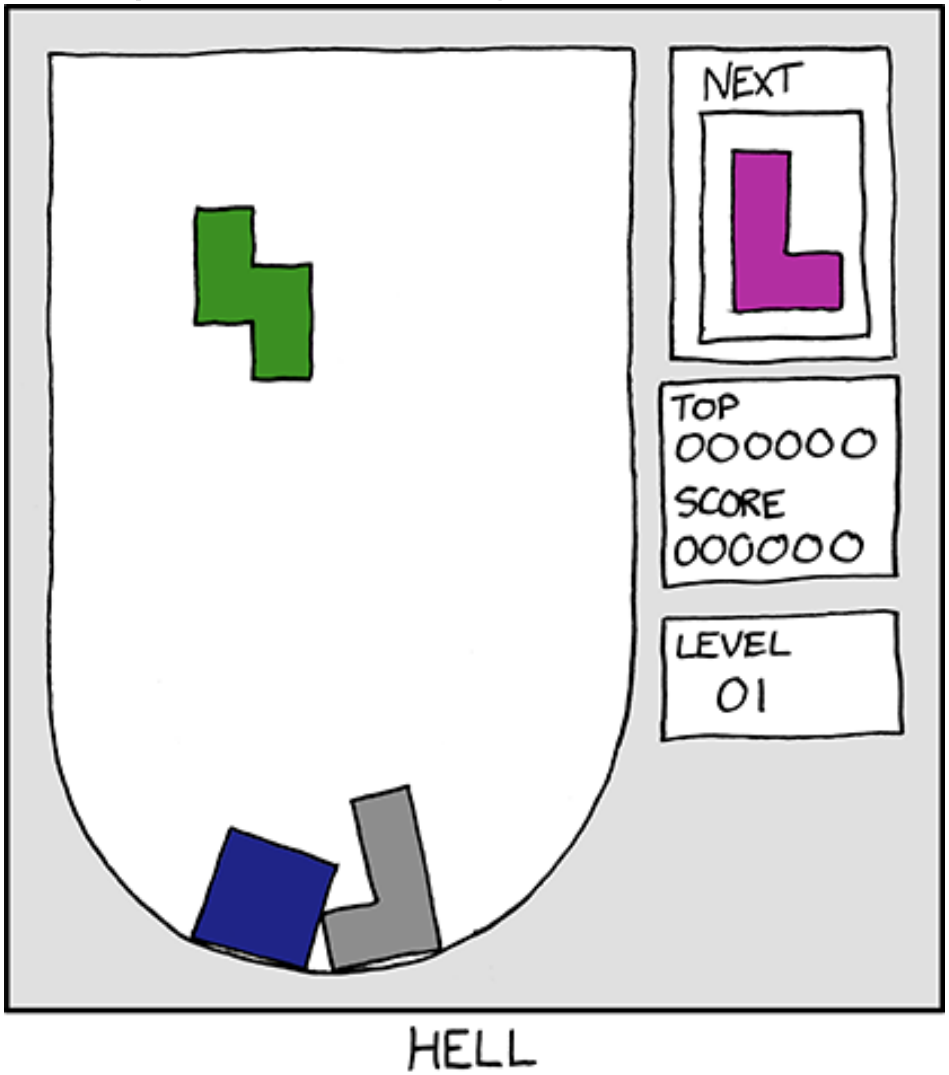
Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.

Modelo de Programación de Aplicaciones



Modelo de Programación de Sistemas

En ocasiones vas a experimentar algo como...



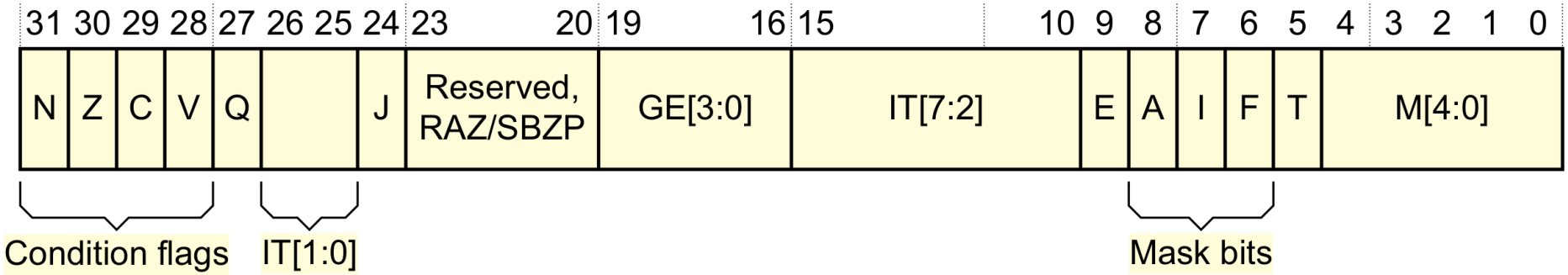
Temario

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **`CPSR`** y **`SPSR`**

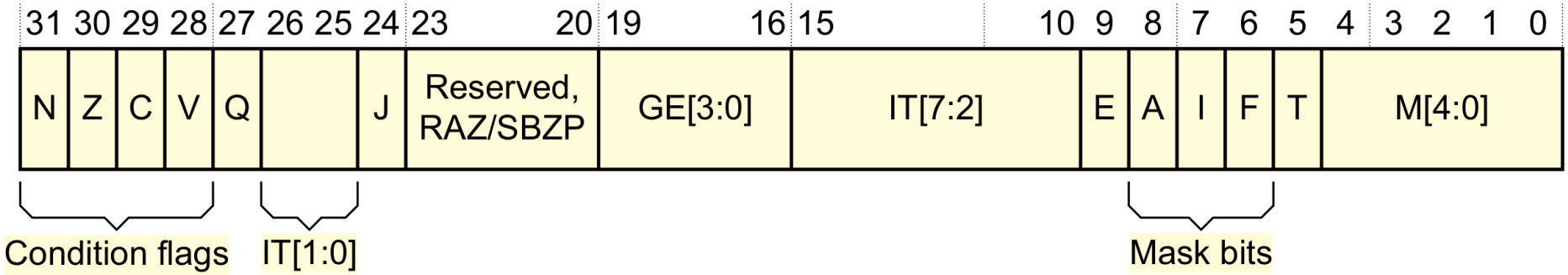
Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**

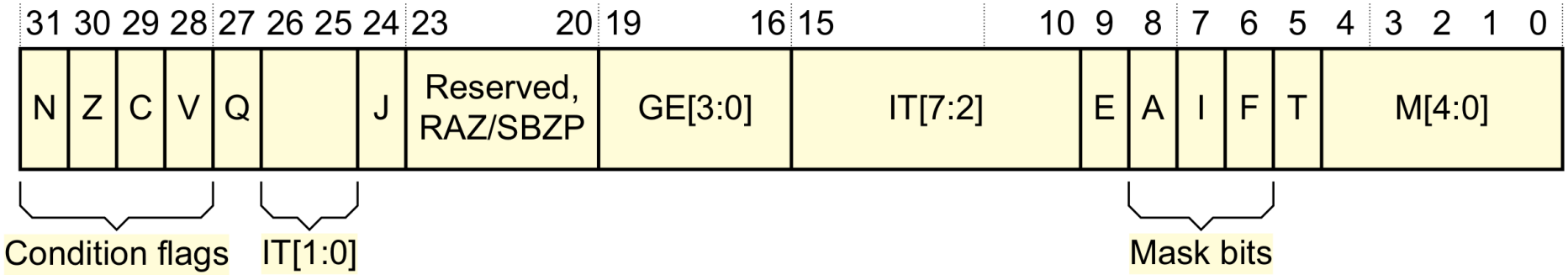


Los bits 24 (**CPSR.J**), y 5 (**CPSR.T**), conforman también un registro denominado **ISETSTATE**, cuyos bits relevantes son los bits 1, y 0.

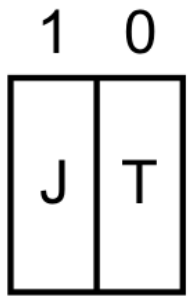
ISETSTATE.J, y **ISETSTATE.T**, son **CPSR.J**, y **CPSR.T**.

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



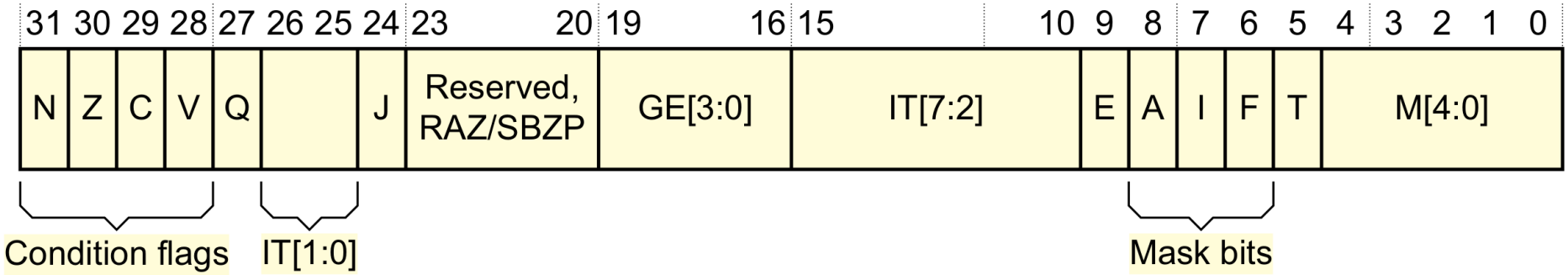
Los bits 24 (**CPSR.J**), y 5 (**CPSR.T**), conforman también un registro denominado **ISETSTATE**, cuyos bits relevantes son los bits 1, y 0.



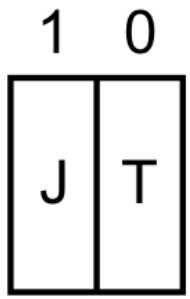
ISETSTATE.J, y **ISETSTATE.T**, son **CPSR.J**, y **CPSR.T**.

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Los bits 24 (**CPSR.J**), y 5 (**CPSR.T**), conforman también un registro denominado **ISETSTATE**, cuyos bits relevantes son los bits 1, y 0.

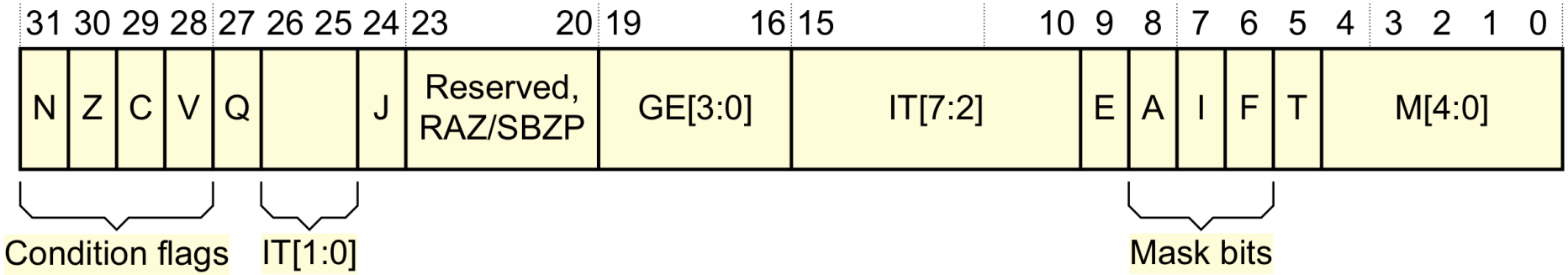


ISETSTATE.J, y **ISETSTATE.T**, son **CPSR.J**, y **CPSR.T**.

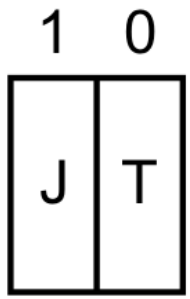
J	T	Instruction set state
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	ThumbEE

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Los bits 24 (**CPSR.J**), y 5 (**CPSR.T**), conforman también un registro denominado **ISETSTATE**, cuyos bits relevantes son los bits 1, y 0.



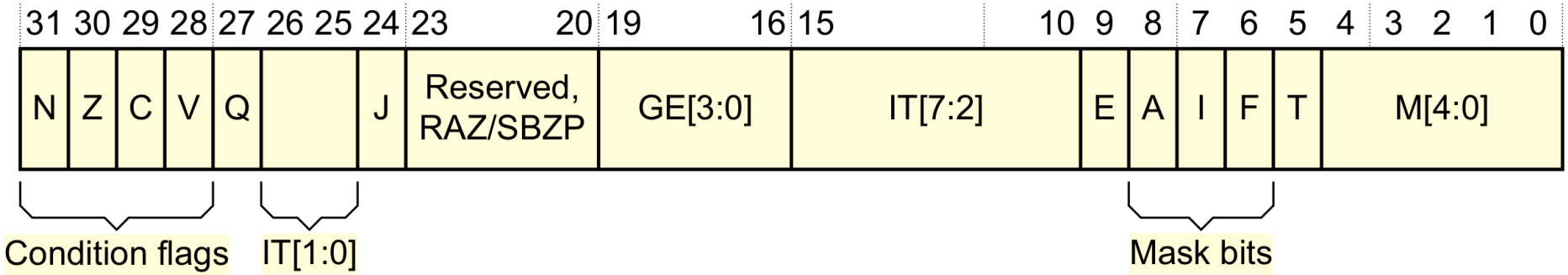
ISETSTATE.J, y **ISETSTATE.T**, son **CPSR.J**, y **CPSR.T**.

J	T	Instruction set state
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	ThumbEE

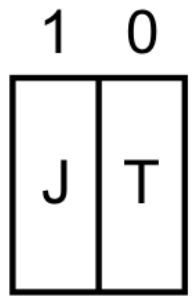
Los modos de interés como ya hemos anticipado en este curso (y que pueden setearse en assembler) son ARM y Thumb, los cuales se seleccionan con el bit **CPSR.T** o **ISETSTATE.T** (según se prefiera).

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Los bits 24 (**CPSR.J**), y 5 (**CPSR.T**), conforman también un registro denominado **ISETSTATE**, cuyos bits relevantes son los bits 1, y 0.



ISETSTATE.J, y **ISETSTATE.T**, son **CPSR.J**, y **CPSR.T**.

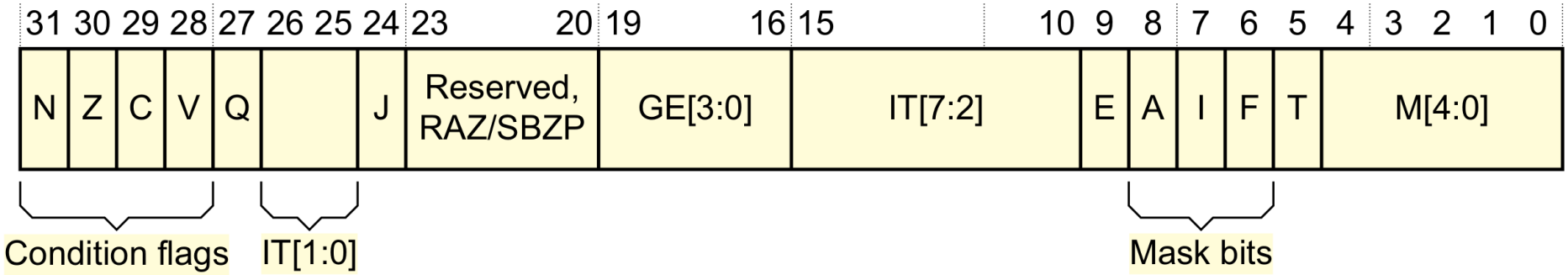
J	T	Instruction set state
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	ThumbEE

Los modos de interés como ya hemos anticipado en este curso (y que pueden setearse en assembler) son ARM y Thumb, los cuales se seleccionan con el bit **CPSR.T** o **ISETSTATE.T** (según se prefiera).

```
bx Rm /*Branch with eXchange. El bit Rm.0 define el modo:
        if Rm.0 = 0, set ARM Mode
        if Rm.0 = 1, set Thumb Mode*/
```

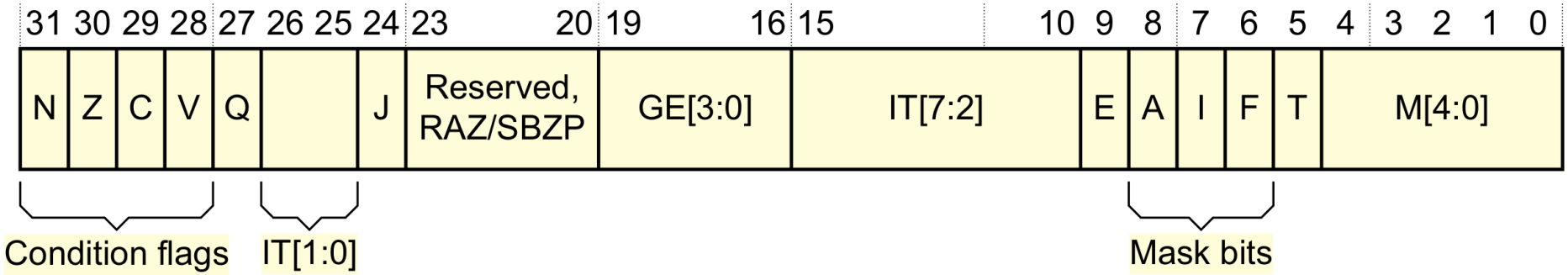
Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Estados, Modos, y Niveles de Privilegio

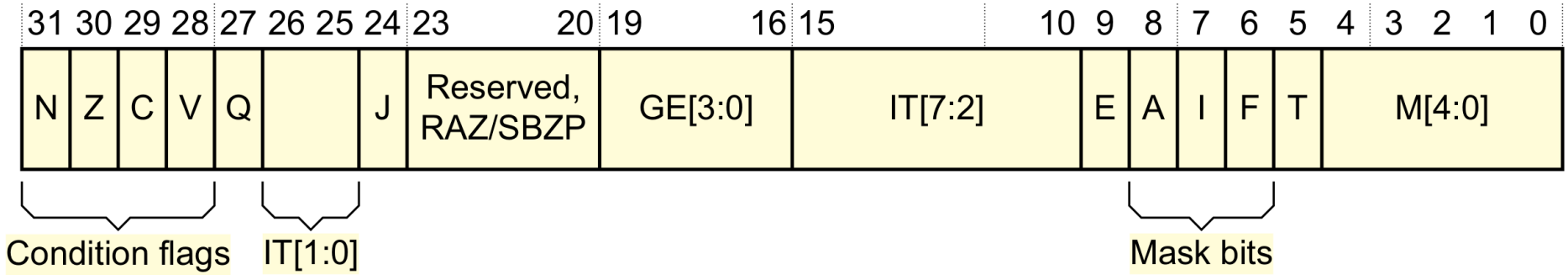
Recordemos el formato del registro **`CPSR`** y **`SPSR`**



Los modos se definen en los bits **`M[4-0]`**.

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**

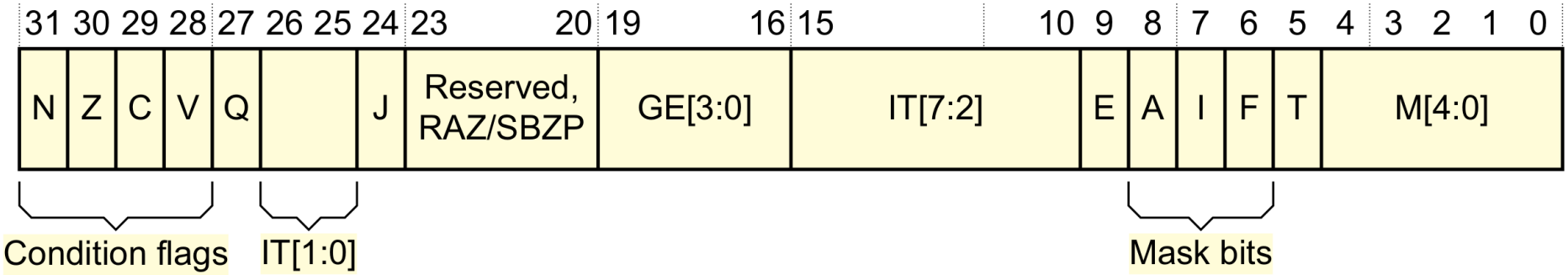


Los modos se definen en los bits **M[4-0]**.

Processor mode		Encoding	Privilege level	Implemented	Security state
User	usr	10000	PL0	Always	Both
FIQ	fiq	10001	PL1	Always	Both
IRQ	irq	10010	PL1	Always	Both
Supervisor	svc	10011	PL1	Always	Both
Monitor	mon	10110	PL1	With Security Extensions	Secure only
Abort	abt	10111	PL1	Always	Both
Hyp	hyp	11010	PL2	With Virtualization Extensions	Non-secure only
Undefined	und	11011	PL1	Always	Both
System	sys	11111	PL1	Always	Both

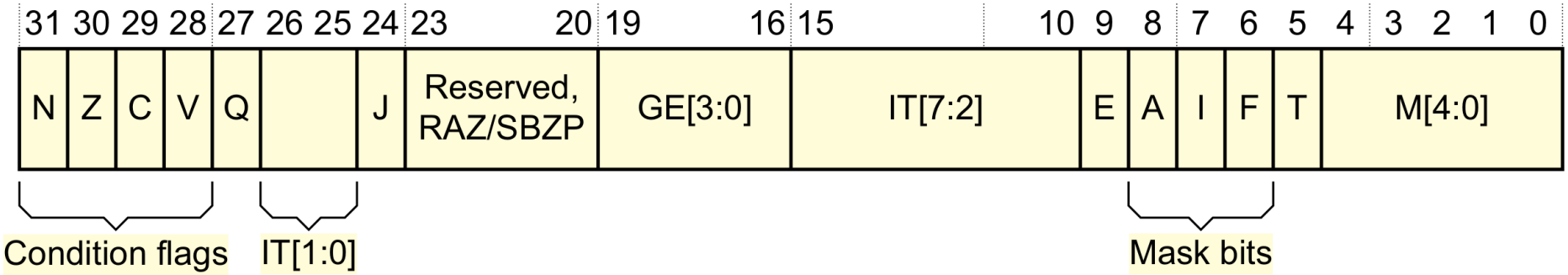
Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**



Estados, Modos, y Niveles de Privilegio

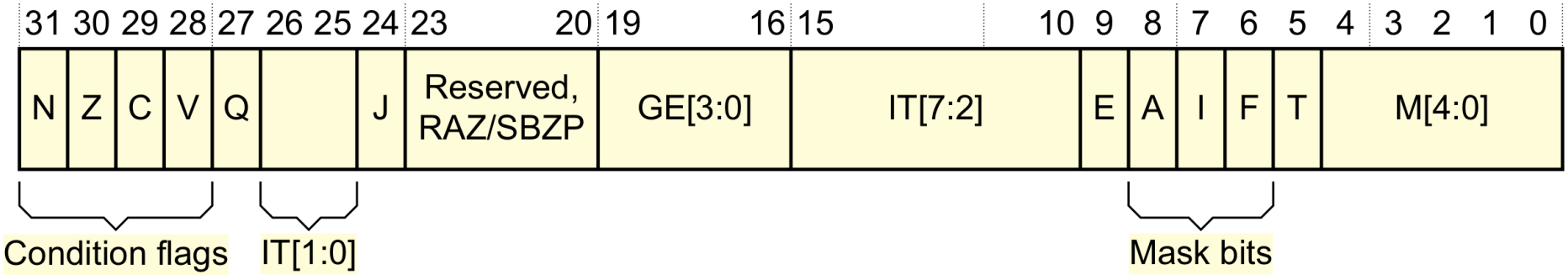
Recordemos el formato del registro **CPSR** y **SPSR**



El estado en un procesador ARM responde a los siguientes conceptos:

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **CPSR** y **SPSR**

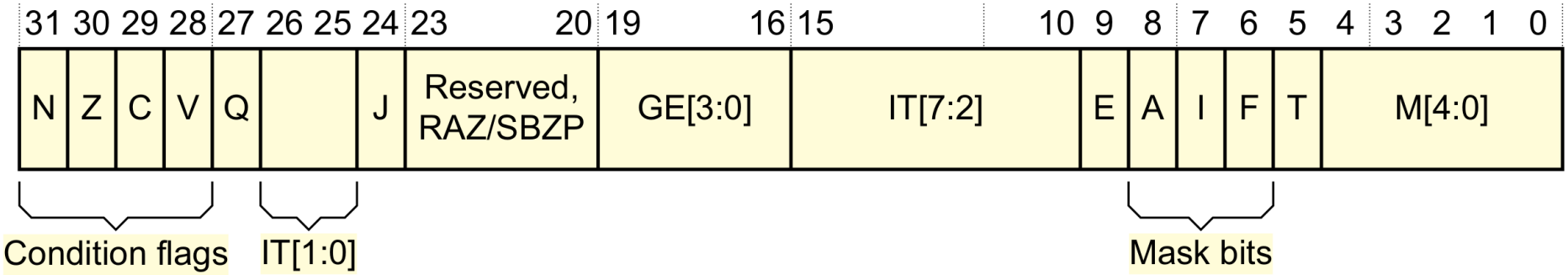


El estado en un procesador ARM responde a los siguientes conceptos:

- Estado Instruction Set: ARM, Thumb, Jazelle, o ThumbEE.

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **`CPSR`** y **`SPSR`**

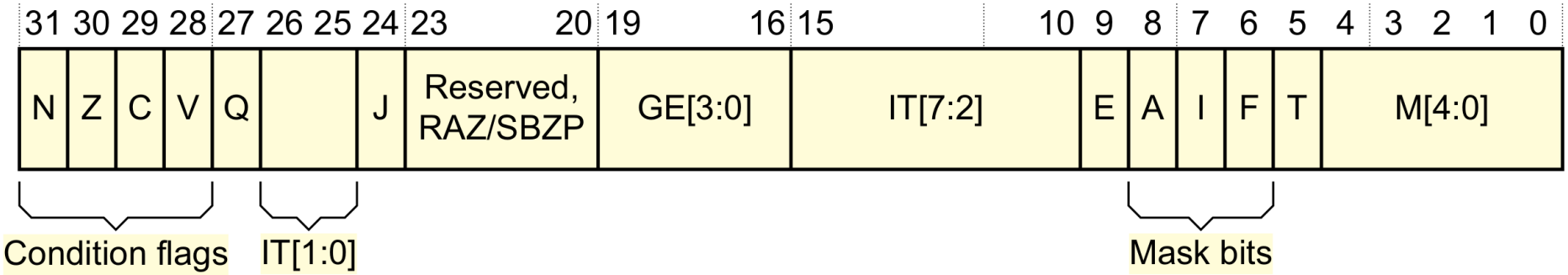


El estado en un procesador ARM responde a los siguientes conceptos:

- Estado Instruction Set: ARM, Thumb, Jazelle, o ThumbEE.
- Estado de ejecución: Intervienen los bits de Modo, **`IT[7:0]`** en Modo Thumb, y el bit de Endianess **`CSPR.E`** (bit 9).

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **`CPSR`** y **`SPSR`**

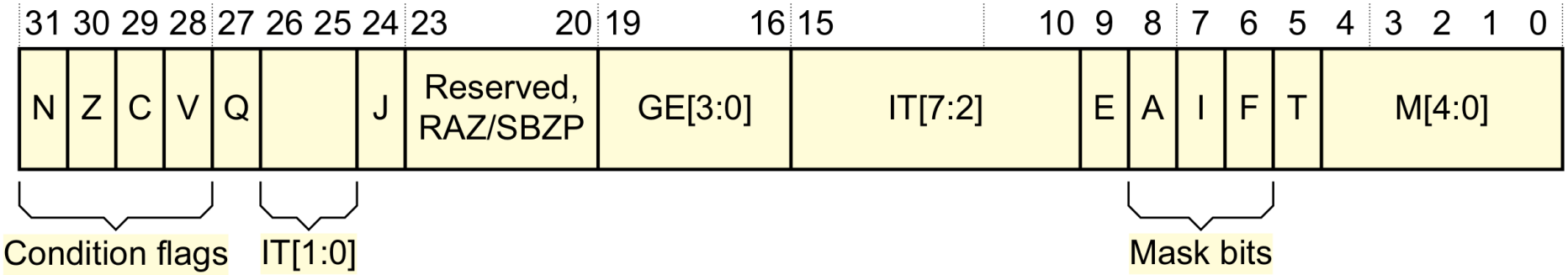


El estado en un procesador ARM responde a los siguientes conceptos:

- Estado Instruction Set: ARM, Thumb, Jazelle, o ThumbEE.
- Estado de ejecución: Intervienen los bits de Modo, **`IT[7:0]`** en Modo Thumb, y el bit de Endianess **`CSPR.E`** (bit 9).
- Estado de Seguridad: Depende de la implementación en el core de la Extensión de seguridad de la ARMv7. Si el core tiene esta extensión y se la habilita, aparecen dos estados: Secure y Non-Secure.

Estados, Modos, y Niveles de Privilegio

Recordemos el formato del registro **`CPSR`** y **`SPSR`**



El estado en un procesador ARM responde a los siguientes conceptos:

- Estado Instruction Set: ARM, Thumb, Jazelle, o ThumbEE.
- Estado de ejecución: Intervienen los bits de Modo, **`IT[7:0]`** en Modo Thumb, y el bit de Endianess **`CSPR.E`** (bit 9).
- Estado de Seguridad: Depende de la implementación en el core de la Extensión de seguridad de la ARMv7. Si el core tiene esta extensión y se la habilita, aparecen dos estados: Secure y Non-Secure.
- Estado de Debug: El procesador está detenido (Halted) con propósito de Debug. Por omisión está en estado No-Debug.

Estados, Modos, y Niveles de Privilegio

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

`PL0`: Ejecución en Modo User

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

`PL0`: Ejecución en Modo User

`PL1`: Ejecución en Modo != User.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

`PL0`: Ejecución en Modo User

`PL1`: Ejecución en Modo != User.

Estado No Seguro : En este estado hay 2 o 3 Niveles de Privilegio.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

`PL0`: Ejecución en Modo User

`PL1`: Ejecución en Modo != User.

Estado No Seguro : En este estado hay 2 o 3 Niveles de Privilegio.

`PL0`: Ejecución en Modo User.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

`PL0`: Ejecución en Modo User

`PL1`: Ejecución en Modo != User.

Estado No Seguro : En este estado hay 2 o 3 Niveles de Privilegio.

`PL0`: Ejecución en Modo User.

`PL1`: Ejecución en Modo != User o != Hyp.

Estados, Modos, y Niveles de Privilegio

- El Nivel de Privilegio es un atributo del software de ejecución.
- Depende del Estado de Ejecución y del Modo de Ejecución

Estado Seguro : En este estado hay dos niveles de Privilegio.

PL0: Ejecución en Modo User

PL1: Ejecución en Modo != User.

Estado No Seguro : En este estado hay 2 o 3 Niveles de Privilegio.

PL0: Ejecución en Modo User.

PL1: Ejecución en Modo != User o != Hyp.

PL2: Ejecución en Modo Hyp (Ext. Virtualización).

Temario

Excepciones

Excepciones

- Ya nos hemos ocupado de los aspectos generales.

Excepciones

- Ya nos hemos ocupado de los aspectos generales.
- Pero a la luz de esta nueva perspectiva de System Programming aparecen otros elementos a considerar.

Excepciones

- Ya nos hemos ocupado de los aspectos generales.
- Pero a la luz de esta nueva perspectiva de System Programming aparecen otros elementos a considerar.
- Al momento de generarse la excepción el sistema está en un estado determinado. Ese estado se presenta al handler de excepción.

Excepciones

- Ya nos hemos ocupado de los aspectos generales.
- Pero a la luz de esta nueva perspectiva de System Programming aparecen otros elementos a considerar.
- Al momento de generarse la excepción el sistema está en un estado determinado. Ese estado se presenta al handler de excepción.
- La arquitectura define en que modo se toma cada excepción, pero si están habilitadas las extensiones de Seguridad, éstas también influyen en el modo de la excepción.

Temario

Modos del procesador ARM

Processor mode		Encoding	Privilege level	Implemented	Security state
User	usr	10000	PL0	Always	Both
FIQ	fiq	10001	PL1	Always	Both
IRQ	irq	10010	PL1	Always	Both
Supervisor	svc	10011	PL1	Always	Both
Monitor	mon	10110	PL1	With Security Extensions	Secure only
Abort	abt	10111	PL1	Always	Both
Hyp	hyp	11010	PL2	With Virtualization Extensions	Non-secure only
Undefined	und	11011	PL1	Always	Both
System	sys	11111	PL1	Always	Both

Modo User

Modo User

- En este modo deben ejecutar las tareas del sistema que no deben tener acceso a los recursos sensibles (Memoria, dispositivos de hardware, etc). Este acceso lo deben solicitar a través del kernel.

Modo User

- En este modo deben ejecutar las tareas del sistema que no deben tener acceso a los recursos sensibles (Memoria, dispositivos de hardware, etc). Este acceso lo deben solicitar a través del kernel.
- Se lo suele llamar Modo No Privilegiado.

Modo User

- En este modo deben ejecutar las tareas del sistema que no deben tener acceso a los recursos sensibles (Memoria, dispositivos de hardware, etc). Este acceso lo deben solicitar a través del kernel.
- Se lo suele llamar Modo No Privilegiado.
- Su nivel de privilegio es **`PL0`**

Modo User

- En este modo deben ejecutar las tareas del sistema que no deben tener acceso a los recursos sensibles (Memoria, dispositivos de hardware, etc). Este acceso lo deben solicitar a través del kernel.
- Se lo suele llamar Modo No Privilegiado.
- Su nivel de privilegio es **`PL0`**
- Un código que ejecuta en este nivel de privilegio no puede cambiar el Modo. Salvo cuando se genera una excepción por supuesto.

Modo User

- En este modo deben ejecutar las tareas del sistema que no deben tener acceso a los recursos sensibles (Memoria, dispositivos de hardware, etc). Este acceso lo deben solicitar a través del kernel.
- Se lo suele llamar Modo No Privilegiado.
- Su nivel de privilegio es **`PL0`**
- Un código que ejecuta en este nivel de privilegio no puede cambiar el Modo. Salvo cuando se genera una excepción por supuesto.

Concepto importante

Las tareas ***deben ejecutar*** en este nivel de privilegio. Aun en un Cortex-M, en donde solo se tiene este modo y el Privilegiado. No hacerlo (es decir poner todo el código en Modo Privilegiado), hace que desarrollar un Sistema Operativo carezca de sentido. Es un diseño básico de microcontroller de la década del 80, aplicado a microcontroladores o microprocesadores modernos.

Modo System

Modo System

- En este modo el código ejecuta en Modo Privilegiado.

Modo System

- En este modo el código ejecuta en Modo Privilegiado.
- Su nivel de privilegio es **`PL1`**.

Modo System

- En este modo el código ejecuta en Modo Privilegiado.
- Su nivel de privilegio es **`PL1`**.
- No se llega a este modo como producto de ningún tipo de excepción.

Modo System

- En este modo el código ejecuta en Modo Privilegiado.
- Su nivel de privilegio es **`PL1`**.
- No se llega a este modo como producto de ningún tipo de excepción.
- Comparte los mismos registros con el modo User.

Modo System

- En este modo el código ejecuta en Modo Privilegiado.
- Su nivel de privilegio es **`PL1`**.
- No se llega a este modo como producto de ningún tipo de excepción.
- Comparte los mismos registros con el modo User.
- Típicamente este es el modo en el que se ejecutan tareas privilegiadas (o sea tareas que son parte del propio kernel). Es un subconjunto de tareas que ejecutan en modo privilegiado y que forman parte del kernel. No componen las tareas de aplicación sino que firman parte de las actividades de administración de los recursos. En Linux se las llama *kernel threads*, solo por citar un ejemplo.

Modo Supervisor

Modo Supervisor

- Supervisor es el modo default que se toma luego de una Supervisor Call exception.

Modo Supervisor

- Supervisor es el modo default que se toma luego de una Supervisor Call exception.
- Ejecutando la instrucción SVC (Supervisor Call) se genera una Supervisor Call exception, que se toma en Modo Supervisor.

Modo Supervisor

- Supervisor es el modo default que se toma luego de una Supervisor Call exception.
- Ejecutando la instrucción SVC (Supervisor Call) se genera una Supervisor Call exception, que se toma en Modo Supervisor.
- Su nivel de privilegio es **`PL1`**.

Modo Supervisor

- Supervisor es el modo default que se toma luego de una Supervisor Call exception.
- Ejecutando la instrucción SVC (Supervisor Call) se genera una Supervisor Call exception, que se toma en Modo Supervisor.
- Su nivel de privilegio es **`PL1`**.
- Luego de un reset el procesador entra en este Modo.

Modo Supervisor

- Supervisor es el modo default que se toma luego de una Supervisor Call exception.
- Ejecutando la instrucción SVC (Supervisor Call) se genera una Supervisor Call exception, que se toma en Modo Supervisor.
- Su nivel de privilegio es **`PL1`**.
- Luego de un reset el procesador entra en este Modo.

Concepto Importante

Un procesador debe arrancar en Modo Privilegiado.
Esto es mas que lógico ya que en este modo hay acceso irrestricto a todo el sistema lo cual es fundamental para construir el firmware básico de test, diagnóstico, e inicialización, y el boot loader del eventual sistema operativo, mas el HAL.

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

- FIQ Modo default cuando se procesa una Fast IRQ

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

- **FIQ** Modo default cuando se procesa una Fast IRQ
- **IRQ** Modo default para interrupciones

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

FIQ Modo default cuando se procesa una Fast IRQ

IRQ Modo default para interrupciones

Abort Modo default cuando se produce error en un Fetch o en un acceso a datos en memoria

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

FIQ Modo default cuando se procesa una Fast IRQ

IRQ Modo default para interrupciones

Abort Modo default cuando se produce error en un Fetch o en un acceso a datos en memoria

Undefined Modo default cuando se encuentra un código de operación inválido

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

FIQ	Modo default cuando se procesa una Fast IRQ
IRQ	Modo default para interrupciones
Abort	Modo default cuando se produce error en un Fetch o en un acceso a datos en memoria
Undefined	Modo default cuando se encuentra un código de operación inválido

- **`FIQ`** e **`IRQ`** están directamente relacionadas con las interrupciones de Hardware y son modos que se definen en cada controlador de interrupciones una a una. Es decir que al ser dependiente del fabricante, hay que mirar en cada procesador el controlador de interrupciones para setear el modo a cara fuente de interrupción.

Mas Modos privilegiados

Todos estos modos ejecutan en **`PL1`**

FIQ	Modo default cuando se procesa una Fast IRQ
IRQ	Modo default para interrupciones
Abort	Modo default cuando se produce error en un Fetch o en un acceso a datos en memoria
Undefined	Modo default cuando se encuentra un código de operación inválido

- **`FIQ`** e **`IRQ`** están directamente relacionadas con las interrupciones de Hardware y son modos que se definen en cada controlador de interrupciones una a una. Es decir que al ser dependiente del fabricante, hay que mirar en cada procesador el controlador de interrupciones para setear el modo a cara fuente de interrupción.
- Los otros dos modos corresponde a las excepciones homónimas. No mas que eso. Son predefinidas por la arquitectura

Modos relacionados con extensiones

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.
- Cuando se activan las extensiones de virtualización se activa un nivel de privilegio mayor que corresponde al Hypervisor de un sistema de virtualización (es decir el sistema anfitrión que permite tener diferentes sistemas operativos en el mismo computador)

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.
- Cuando se activan las extensiones de virtualización se activa un nivel de privilegio mayor que corresponde al Hypervisor de un sistema de virtualización (es decir el sistema anfitrión que permite tener diferentes sistemas operativos en el mismo computador)
- El modo se denomina **`Hyp`**, y su nivel de privilegio es **`PL2`**.

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.
- Cuando se activan las extensiones de virtualización se activa un nivel de privilegio mayor que corresponde al Hypervisor de un sistema de virtualización (es decir el sistema anfitrión que permite tener diferentes sistemas operativos en el mismo computador)
- El modo se denomina **`Hyp`**, y su nivel de privilegio es **`PL2`**.
- Virtualización siempre trabaja en un estado **`Non-Secure`**.

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.
- Cuando se activan las extensiones de virtualización se activa un nivel de privilegio mayor que corresponde al Hypervisor de un sistema de virtualización (es decir el sistema anfitrión que permite tener diferentes sistemas operativos en el mismo computador)
- El modo se denomina **`Hyp`**, y su nivel de privilegio es **`PL2`**.
- Virtualización siempre trabaja en un estado **`Non-Secure`**.
- Se ingresa al modo **`Hyp`** mediante la instrucción HVC (Hyper Visor Call) ejecutada en código **`Non-Secure`** con **`PL1`**, lo que genera una Hyper Visor Call exception.

Modos relacionados con extensiones

- Aquí solo a modo referencia mencionamos las extensiones *Security* y *Virtualization*.
- Cuando se activan las extensiones de virtualización se activa un nivel de privilegio mayor que corresponde al Hypervisor de un sistema de virtualización (es decir el sistema anfitrión que permite tener diferentes sistemas operativos en el mismo computador)
- El modo se denomina **`Hyp`**, y su nivel de privilegio es **`PL2`**.
- Virtualización siempre trabaja en un estado **`Non-Secure`**.
- Se ingresa al modo **`Hyp`** mediante la instrucción HVC (Hyper Visor Call) ejecutada en código **`Non-Secure`** con **`PL1`**, lo que genera una Hyper Visor Call exception.
- Otra forma es mediante una Hyper Visor Trap Exception.

Modos relacionados con extensiones

Modos relacionados con extensiones

- Cuando activamos las extensiones de Seguridad aparece el modo **`Monitor`** al que se accede vía la Secure Monitor Call Exception.

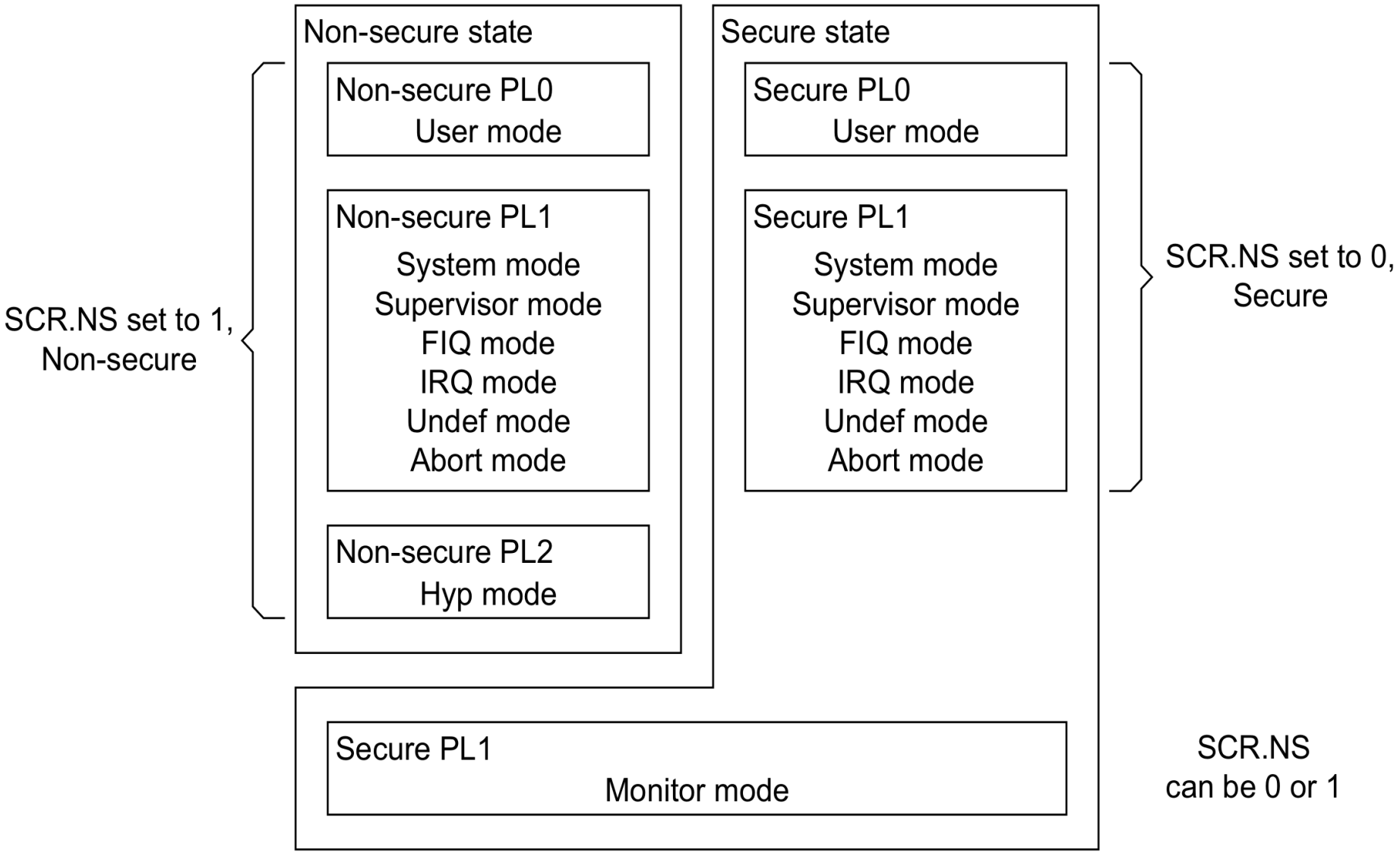
Modos relacionados con extensiones

- Cuando activamos las extensiones de Seguridad aparece el modo **`Monitor`** al que se accede vía la Secure Monitor Call Exception.
- Esto se puede hacer ejecutando la instrucción SMC (Secure Monitor Call) desde código en **`PL1`**.Cualquier código ejecutando en Modo Monitor tiene acceso a las copia de los registros **`Secure`** o **`Non-Secure`**.

Modos relacionados con extensiones

- Cuando activamos las extensiones de Seguridad aparece el modo **`Monitor`** al que se accede vía la Secure Monitor Call Exception.
- Esto se puede hacer ejecutando la instrucción SMC (Secure Monitor Call) desde código en **`PL1`**.Cualquier código ejecutando en Modo Monitor tiene acceso a las copia de los registros **`Secure`** o **`Non-Secure`**.
- Con estas extensiones activas cada procesador está en un estado de los que conocemos pero **`Secure`** o **`Non-Secure`**. Por ejemplo, **`Secure Supervisor Mode`**.

Modos y Privilegios y estados



Modos y Registros

Application level view	System level view								
	User	System	Hyp †	Supervisor	Abort	Undefined	Monitor ‡	IRQ	FIQ
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp	SP_svc	SP_abt	SP_und	SP_mon	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_abt	LR_und	LR_mon	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon	SPSR_irq	SPSR_fiq
			ELR_hyp						

Temario

Soporte a Coprocesadores

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.
- Mediante un puñado de instrucciones especiales de acceso soporta hasta 16 coprocesadores: `CP0` a `CP15`.

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.
- Mediante un puñado de instrucciones especiales de acceso soporta hasta 16 coprocesadores: `CP0` a `CP15`.
- Los primeros 8, `CP0` a `CP7` no serán empleados por ARM y eventualmente quedan para que cada vendedor provea a través de ellos funcionalidades específicas.

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.
- Mediante un puñado de instrucciones especiales de acceso soporta hasta 16 coprocesadores: **`CP0`** a **`CP15`**.
- Los primeros 8, **`CP0`** a **`CP7`** no serán empleados por ARM y eventualmente quedan para que cada vendedor provea a través de ellos funcionalidades específicas.
- **`CP14`** y **`CP15`**, son corpocesadores que prestan funciones de control sobre la arquitectura y sobre el resto de los coprocesadores.

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.
- Mediante un puñado de instrucciones especiales de acceso soporta hasta 16 coprocesadores: **`CP0`** a **`CP15`**.
- Los primeros 8, **`CP0`** a **`CP7`** no serán empleados por ARM y eventualmente quedan para que cada vendedor provea a través de ellos funcionalidades específicas.
- **`CP14`** y **`CP15`**, son corpocesadores que prestan funciones de control sobre la arquitectura y sobre el resto de los coprocesadores.
- **`CP10`** y **`CP11`**, controlan en forma conjunta las operaciones de punto flotante, vectoriales, y activan configuran y controlan las Unidades de Punto Flotante y la Unidad de SIMD Advanzada.

Soporte a Coprocesadores

- ARMv7 extiende las funcionalidades del procesador mediante la definición de coprocesadores.
- Mediante un puñado de instrucciones especiales de acceso soporta hasta 16 coprocesadores: **`CP0`** a **`CP15`**.
- Los primeros 8, **`CP0`** a **`CP7`** no serán empleados por ARM y eventualmente quedan para que cada vendedor provea a través de ellos funcionalidades específicas.
- **`CP14`** y **`CP15`**, son corpocesadores que prestan funciones de control sobre la arquitectura y sobre el resto de los coprocesadores.
- **`CP10`** y **`CP11`**, controlan en forma conjunta las operaciones de punto flotante, vectoriales, y activan configuran y controlan las Unidades de Punto Flotante y la Unidad de SIMD Advanzada.
- Los **`CP8`**, **`CP9`**, **`CP12`**, y **`CP13`**, están reservados por ARM para utilizarlos en futuras eventuales extensiones. Cualquier intento de acceso a los mismos dará como resultado Undefined Exception.

Coprocesadores de control

Coprocesadores de control

- **`CP14`** y **`CP15`**, son dos coprocesadores imprescindibles en el desarrollo de un sistema multitarea.

Coprocesadores de control

- **`CP14`** y **`CP15`**, son dos coprocesadores imprescindibles en el desarrollo de un sistema multitarea.
- **`CP15`**, permite activar, configurar y controlar la Unidad de Gestión de Memoria (**MMU**, **M**emory **M**anagement **U**nit) en sus dos modalidades posibles: VMSA o PMSA, además de acceder a registros de Monitoreo de Performance

Coprocesadores de control

- **`CP14`** y **`CP15`**, son dos coprocesadores imprescindibles en el desarrollo de un sistema multitarea.
- **`CP15`**, permite activar, configurar y controlar la Unidad de Gestión de Memoria (**MMU**, **M**emory **M**anagement **U**nit) en sus dos modalidades posibles: VMSA o PMSA, además de acceder a registros de Monitoreo de Performance
- **`CP14`**, gestiona los registros de Debug, Trace, y entornos de ejecución (Jazelle y Thumbee)

Coprocesadores de control

- **`CP14`** y **`CP15`**, son dos coprocesadores imprescindibles en el desarrollo de un sistema multitarea.
- **`CP15`**, permite activar, configurar y controlar la Unidad de Gestión de Memoria (**MMU**, **M**emory **M**anagement **U**nit) en sus dos modalidades posibles: VMSA o PMSA, además de acceder a registros de Monitoreo de Performance
- **`CP14`**, gestiona los registros de Debug, Trace, y entornos de ejecución (Jazelle y Thumbee)
- En general se acceden desde **`PL1`**, aunque determinados accesos pueden efectuarse desde **`PL0`**, de modo de proveer acceso controlado a memoria y debug directo desde una tarea.

Registros de control de CP14 y CP15

Registros de control de CP14 y CP15

- Los registros del CP15 varían fuertemente si el procesador es CORTEX-A o si es CORTEX-R

Registros de control de CP14 y CP15

- Los registros del CP15 varían fuertemente si el procesador es CORTEX-A o si es CORTEX-R
- En el caso de los CORTEX-A la MMU habilita el soporte para una MMU que implementa ***Virtual Mode System Architecture*** (***VMSA***) administrando la memoria por un sistema de traducción basado en paginación del espacio de direccionamiento.

Registros de control de CP14 y CP15

- Los registros del CP15 varían fuertemente si el procesador es CORTEX-A o si es CORTEX-R
- En el caso de los CORTEX-A la MMU habilita el soporte para una MMU que implementa ***Virtual Mode System Architecture*** (***VMSA***) administrando la memoria por un sistema de traducción basado en paginación del espacio de direccionamiento.
- En el caso de los CORTEX-R la MMU habilita el soporte para una MMU que implementa ***Protected Mode System Architecture*** (***PMSA***) administrando la memoria asignando diferentes partes del espacio de direccionamiento a cada tarea y al Sistema Operativo.

Registros de control de CP14 y CP15

En la documentación podemos leer que el CORTEX-A tiene MMU (Memory Management Unit), y el R tiene en cambio MPU(Memory Protection Unit). Esta nomenclatura pertenece a ARM. Lo cierto es que en ambos casos la unidad que administra la generación de direcciones y acceso a memoria se llama Memory Management Unit. Y en ambos casos se provee un sistema de Protección para la memoria de cada tarea.

Organización de los registros en CP14 y CP15

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {`CRn,opc1,CRm,opc2`}, establece el orden de los registros de un coprocesador.

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {`CRn,opc1,CRm,opc2`}, establece el orden de los registros de un coprocesador.
- Estos cuatro valores ordenados forman parte de los operandos de las instrucciones, aunque no van en este orden en la instrucción.

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {`CRn,opc1,CRm,opc2`}, establece el orden de los registros de un coprocesador.
- Estos cuatro valores ordenados forman parte de los operandos de las instrucciones, aunque no van en este orden en la instrucción.
- Si me permiten el reduccionismo, CP14 y CP15 son un gran arreglo de registros de control.

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {**CRn, opc1, CRm, opc2**}, establece el orden de los registros de un coprocesador.
- Estos cuatro valores ordenados forman parte de los operandos de las instrucciones, aunque no van en este orden en la instrucción.
- Si me permiten el reduccionismo, CP14 y CP15 son un gran arreglo de registros de control.
- Además al ser CORTEX-A y CORTEX-R diferentes en su Unidad de Manejo de Memoria (MMU) la vista de CP15 en particular es diferente de acuerdo al modelo de memoria del Procesador.

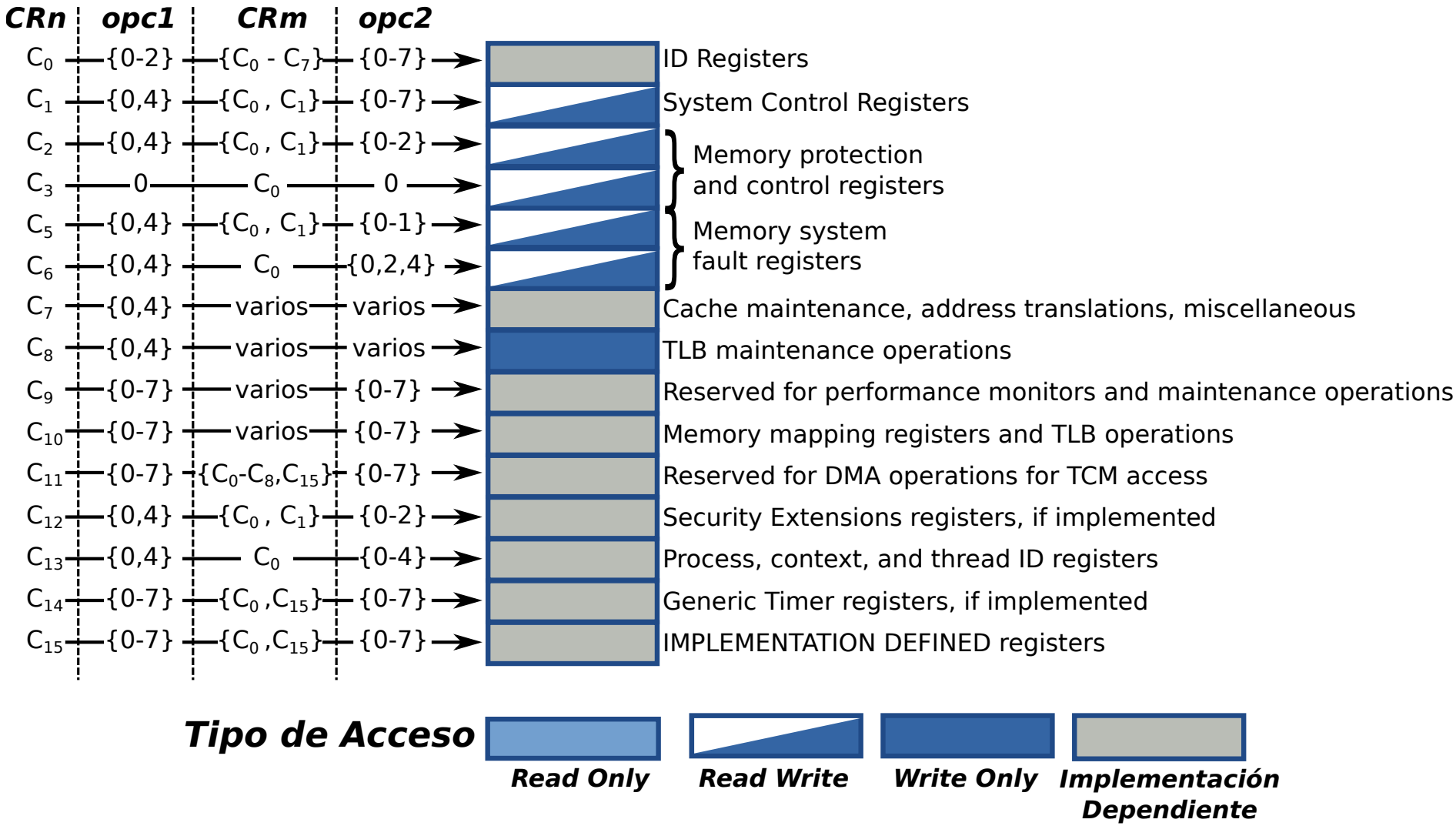
Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {**CRn, opc1, CRm, opc2**}, establece el orden de los registros de un coprocesador.
- Estos cuatro valores ordenados forman parte de los operandos de las instrucciones, aunque no van en este orden en la instrucción.
- Si me permiten el reduccionismo, CP14 y CP15 son un gran arreglo de registros de control.
- Además al ser CORTEX-A y CORTEX-R diferentes en su Unidad de Manejo de Memoria (MMU) la vista de CP15 en particular es diferente de acuerdo al modelo de memoria del Procesador.
- Al estar trabajando con un CORTEX-A8 nos concentraremos en la vista del CP15 para el modelo de memoria VMSA (Virtual Memory System Architecture).

Organización de los registros en CP14 y CP15

- Es uno de los puntos mas oscuros de la floja documentación de System Programming presente en el manual de arquitectura.
- En general el set de valores ordenados {`CRn,opc1,CRm,opc2`}, establece el orden de los registros de un coprocesador.
- Estos cuatro valores ordenados forman parte de los operandos de las instrucciones, aunque no van en este orden en la instrucción.
- Si me permiten el reduccionismo, CP14 y CP15 son un gran arreglo de registros de control.
- Además al ser CORTEX-A y CORTEX-R diferentes en su Unidad de Manejo de Memoria (MMU) la vista de CP15 en particular es diferente de acuerdo al modelo de memoria del Procesador.
- Al estar trabajando con un CORTEX-A8 nos concentraremos en la vista del CP15 para el modelo de memoria VMSA (Virtual Memory System Architecture).
- No obstante haremos especial incapié en los aspectos de interés a los efectos del presente curso.

Vista de CP15 en modo VMSA (CORTEX-A)



Resumen de Registros de CP15 en CORTEX-A

El CP15 se compone de 160 registros. El mecanismo de acceso es a través de cluster de 4 valores {`CRn, opc1, CRm, opc2`}. Combinados adecuadamente, permiten leer o escribir el registro adecuado.

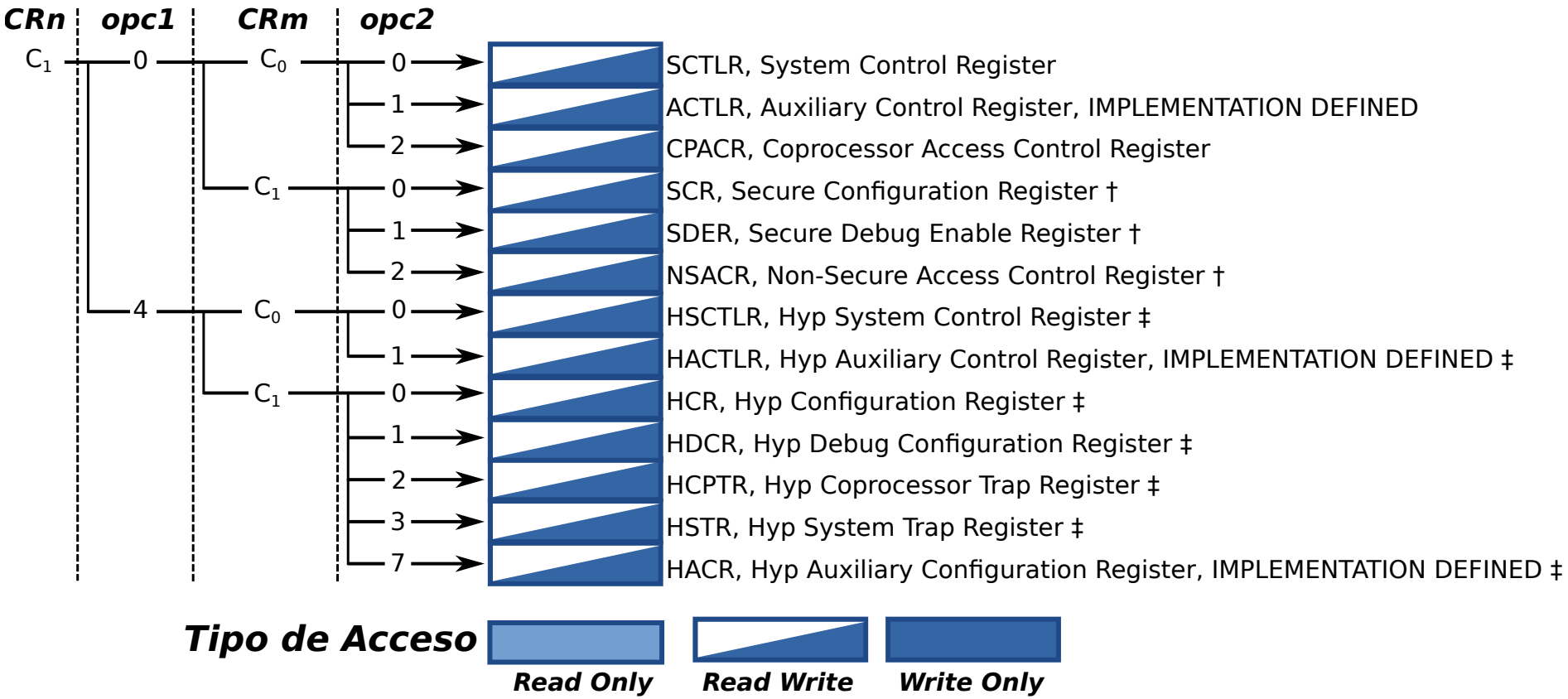
CRn	# Registers	Tipo
c0	27	ID Registers
c1	13	System Control Registers
c2	9	Memory protection and control registers
c3	1	Memory protection and control registers
c5	7	Memory system fault registers
c6	5	Memory system fault registers
c7	28	Cache maintenance, address translations, miscellaneous
c8	20	TLB maintenance operations
c9	15	Reserved for performance monitors and maintenance operations
c10	10	Memory mapping registers and TLB operations
c11	-	Reserved for DMA operations for TCM access
c12	4	Security Extensions registers, if implemented
c13	6	Process, context, and thread ID registers
c14	15	Generic Timer registers, if implemented
c15	-	IMPLEMENTATION DEFINED registers

Cortex-A CP15: CRn=c1 System Control Registers

En la vista general de registros vimos que se accede a los Registros de Control del Sistema mediante el Registro primario C1. El set de valores ordenados {**CRn,opc1,CRm,opc2**} ={c1,{0,4},{c0,c1},{0-7}}.

Cortex-A CP15: CRn=c1 System Control Registers

En la vista general de registros vimos que se accede a los Registros de Control del Sistema mediante el Registro primario C1. El set de valores ordenados {**CRn,opc1,CRm,opc2**} ={c1,{0,4},{c0,c1},{0-7}}.



† Implementado como parte de las Security Extensions solamente
‡ Implementado como parte de las Virtualization Extensions solamente

CPACR, Coprocessor Access Control Register, VMSA

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASEDIS	D32DIS	0	TRCDIS	cp13		cp12		cp11		cp10		cp9		cp8		cp7		cp6		cp5		cp4		cp3		cp2		cp1		cp0	

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASEDIS	D32DIS	0	TRCDIS	cp13		cp12		cp11		cp10		cp9		cp8		cp7		cp6		cp5		cp4		cp3		cp2		cp1		cp0	

ASEDIS ’1’: Deshabilita funcionalidad SIMD Avanzada (cualquier instrucción SIMD genera Unidefined Exception).

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.

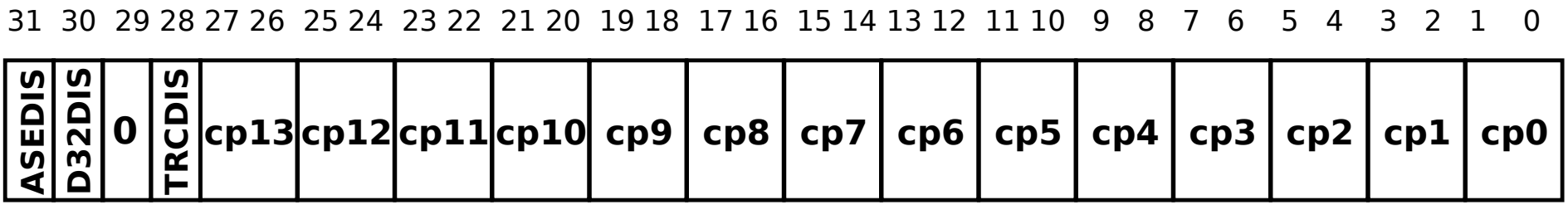
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASEDIS	D32DIS	0	TRCDIS	cp13		cp12		cp11		cp10		cp9		cp8		cp7		cp6		cp5		cp4		cp3		cp2		cp1		cp0	

ASEDIS ’1’: Deshabilita funcionalidad SIMD Avanzada (cualquier instrucción SIMD genera Unidefined Exception).

D32DIS ’1’: Deshabilita los registros D31-D16 de la Unidad de punto flotante.

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.



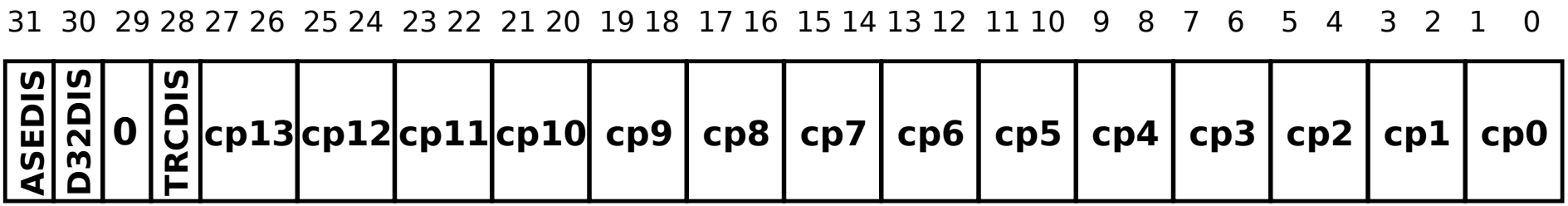
ASEDIS ’1’: Deshabilita funcionalidad SIMD Avanzada (cualquier instrucción SIMD genera Unidefined Exception).

D32DIS ’1’: Deshabilita los registros D31-D16 de la Unidad de punto flotante.

TRCDIS ’1’: Deshabilita el acceso a los Trace Registers del cp14.

CPACR, Coprocessor Access Control Register, VMSA

- Controla el acceso a los coprocesadores cp0 a cp13.
- Accesible solo desde **`PL1`** o mayor.
- Si se habilita la extensión de Virtualización, no tiene efecto si la instrucción que lo accede ejecuta en modo **Hyp**.



ASEDIS '1': Deshabilita funcionalidad SIMD Avanzada (cualquier instrucción SIMD genera Unidefined Exception).

D32DIS '1': Deshabilita los registros D31-D16 de la Unidad de punto flotante.

TRCDIS '1': Deshabilita el acceso a los Trace Registers del cp14.

cpn
00 Acceso denegado. Genera Undefined Instruction Exception.
01 Acceso solo desde **`PL1`**.
10 Reservado. Acceso con resultado impredecible.
11 Acceso irrestricto.

Instrucciones de acceso a los coprocesadores

Hay tres grupos de instrucciones para manejo de los coprocesadores (todas ejecutan en Modo Privilegiado):

Instrucciones de acceso a los coprocesadores

Hay tres grupos de instrucciones para manejo de los coprocesadores (todas ejecutan en Modo Privilegiado):

- Iniciar una operación de datos en un coprocesador.

Instrucciones de acceso a los coprocesadores

Hay tres grupos de instrucciones para manejo de los coprocesadores (todas ejecutan en Modo Privilegiado):

- Iniciar una operación de datos en un coprocesador.
- Transferir Datos entre un registro Core hacia / desde un registro de coprocesador (instrucciones **`MRC`** y **`MCR`** respectivamente)

Instrucciones de acceso a los coprocesadores

Hay tres grupos de instrucciones para manejo de los coprocesadores (todas ejecutan en Modo Privilegiado):

- Iniciar una operación de datos en un coprocesador.
- Transferir Datos entre un registro Core hacia / desde un registro de coprocesador (instrucciones **`MRC`** y **`MCR`** respectivamente)
- Load y Store hacia / desde registros de coprocesador

Ejemplo de acceso a los coprocesadores

Ejemplo de acceso a los coprocesadores

Antes de pasar a la sintaxis de las instrucciones un ejemplo práctico de utilidad.
Se trata de habilitar los coprocesadores de punto flotante y SIMD, **`cp10`**, y **`cp11`** respectivamente.

Ejemplo de acceso a los coprocesadores

Antes de pasar a la sintaxis de las instrucciones un ejemplo práctico de utilidad.

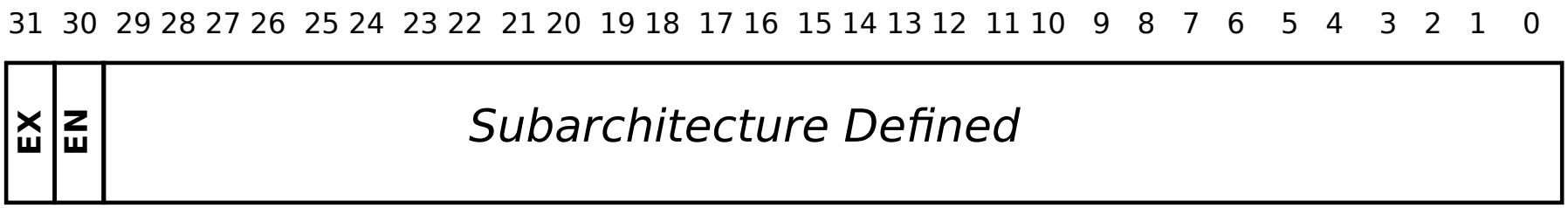
Se trata de habilitar los coprocesadores de punto flotante y SIMD, **`cp10`**, y **`cp11`** respectivamente.

```
1 MRC   p15,0,r0,c1,c0,2  ;Lee registro CPACR en r0
2 ORR   r0,r0,#(3<<20)    ;OR con 0x300000: habilita CP10 para PL1 y PL0
3 ORR   r0,r0,#(3<<22)    ;OR con 0xc00000: habilita CP11 para PL1 y PL0
4 BIC   r0, r0, #(3<<30)  ;Limpia ASEDIS/D32DIS if set
5 MCR   p15,0,r0,c1,c0,2  ;Escribe en CPACR lso nuevos permisos
6 ISB                     ;Flush Pipeline
7 MOV   r0,#(1<<30)       ;Crea mascara con FPEXC (bit 30) '1' en r0
8 VMSR FPEXC,r0           ;Habilita extensiones VFP y SIMD
```

Ejemplo de acceso a los coprocesadores

Antes de pasar a la sintaxis de las instrucciones un ejemplo práctico de utilidad.
Se trata de habilitar los coprocesadores de punto flotante y SIMD, **`cp10`**, y **`cp11`** respectivamente.

```
1 MRC   p15,0,r0,c1,c0,2  ;Lee registro CPACR en r0
2 ORR   r0,r0,#(3<<20)    ;OR con 0x300000: habilita CP10 para PL1 y PL0
3 ORR   r0,r0,#(3<<22)    ;OR con 0xc00000: habilita CP11 para PL1 y PL0
4 BIC   r0, r0, #(3<<30)  ;Limpia ASEDIS/D32DIS if set
5 MCR   p15,0,r0,c1,c0,2  ;Escribe en CPACR lso nuevos permisos
6 ISB                     ;Flush Pipeline
7 MOV   r0,#(1<<30)       ;Crea mascara con FPEXC (bit 30) '1' en r0
8 VMSR FPEXC,r0           ;Habilita extensiones VFP y SIMD
```



Registro de control de la Unidad de PF y SIMD FPEXC.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c código condicional (En T2 no aplica)

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c código condicional (En T2 no aplica)
- q **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c código condicional (En T2 no aplica)
- q **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c código condicional (En T2 no aplica)
- q **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.

coproc Nombre del coprocessor (p0-p15). Es el campo **`cp_num`** del OPCODE.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c: código condicional (En T2 no aplica)
- q: **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.
- coproc: Nombre del coprocessor (p0-p15). Es el campo **cp_num** del OPCODE.
- opc1: Opcode específico del coprocessor (0 a 15).

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c: código condicional (En T2 no aplica)
- q: **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.
- coproc: Nombre del coprocessor (p0-p15). Es el campo **cp_num** del OPCODE.
- opc1: Opcode específico del coprocessor (0 a 15).
- CRd: Registro destino del coprocesador.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c: código condicional (En T2 no aplica)
- q: **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.
- coproc: Nombre del coprocessor (p0-p15). Es el campo **`cp_num`** del OPCODE.
- opc1: Opcode específico del coprocessor (0 a 15).
- CRd: Registro destino del coprocesador.
- CRn: Registro del coprocesador que contiene operando 1.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c: código condicional (En T2 no aplica)
- q: **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.

  .W Wide, usar OPCODE de 32-bit o error si no es posible.
- coproc: Nombre del coprocessor (p0-p15). Es el campo **`cp_num`** del OPCODE.
- opc1: Opcode específico del coprocessor (0 a 15).
- CRd: Registro destino del coprocesador.
- CRn: Registro del coprocesador que contiene operando 1.
- CRm: Registro del coprocesador que contiene operando 2.

CPD{2}Coprocessor Data Processing

Sintaxis:

```
1 CDP  {c}{q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
2 CDP2 {q} coproc, {#}opc1, CRd, CRn, CRm {, {#}opc2}
```

Indica al coprocesador que realice una operación en sus registros y/o memoria independientes del core ARM.

- c: código condicional (En T2 no aplica)
- q: **.N N**arrow, usar OPCODE de 16-bit o error si no es posible.
  .W Wide, usar OPCODE de 32-bit o error si no es posible.
- coproc: Nombre del coprocessor (p0-p15). Es el campo **cp_num** del OPCODE.
- opc1: Opcode específico del coprocessor (0 a 15).
- CRd: Registro destino del coprocesador.
- CRn: Registro del coprocesador que contiene operando 1.
- CRm: Registro del coprocesador que contiene operando 2.
- opc2: Opcode específico del coprocesador (0 a 7). Si se omite, se asume 0.

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond código condicional (En T2 no aplica)

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond código condicional (En T2 no aplica)

coproc Nombre del coprocessor (p0-p15).

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocessor (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocessor (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocessor (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocessor (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).
CRn	Registro 1 del coprocesador.

MRC{2} Mover a un Reg Core desde el Coprocesador

Sintaxis:

```
1 MRC  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MRC2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocessor (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).
CRn	Registro 1 del coprocesador.
CRm	Registro 2 del coprocesador.

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond código condicional (En T2 no aplica)

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond código condicional (En T2 no aplica)

coproc Nombre del coprocesador (p0-p15).

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).
CRn	Registro 1 del coprocesador.

MCR{2} Mover desde un Reg Core al Coprocesador

Sintaxis:

```
1 MCR  {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
2 MCR2 {cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode1	Opcode específico del coprocesador (0 a 7).
opcode2	Opcode específico del coprocesador opcional (0 a 7).
Rt	Registro core ARM (Cualquiera excepto R15).
CRn	Registro 1 del coprocesador.
CRm	Registro 2 del coprocesador.

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond código condicional (En T2 no aplica)

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond código condicional (En T2 no aplica)

coproc Nombre del coprocesador (p0-p15).

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).
Rt2	General Propose Reg. ARM (Cualquiera excepto R15).

MRRC{2} Mover desde el Coprocesador a dos GPR

Sintaxis:

```
1 MRRC  {cond} coproc, #opcode, Rt, Rt2, CRm
2 MRRC2 {cond} coproc, #opcode, Rt, Rt2, CRm
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).
Rt2	General Propose Reg. ARM (Cualquiera excepto R15).
CRm	Registro del coprocesador.

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond código condicional (En T2 no aplica)

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond código condicional (En T2 no aplica)

coproc Nombre del coprocesador (p0-p15).

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).
Rt2	General Propose Reg. ARM (Cualquiera excepto R15).

MCRR{2} Mover desde dos GPR al Coprocesador

Sintaxis:

```
1 MCRR  {cond} coproc, #opcode, Rt, Rt2, CRn
2 MCRR2 {cond} coproc, #opcode, Rt, Rt2, CRn
```

cond	código condicional (En T2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
opcode	Opcode específico del coprocesador (0 a 15).
Rt	General Propose Reg. ARM (Cualquiera excepto R15).
Rt2	General Propose Reg. ARM (Cualquiera excepto R15).
CRm	Registro del coprocesador.

LDC{2}Transfiere de Memoria al Coprocesador

Sintaxis:

```
1 op{L}{cond} coproc, CRd, [Rn]
2 op{L}{cond} coproc, CRd, [Rn, #{-}offset] ; direc. por offset
3 op{L}{cond} coproc, CRd, [Rn, #{-}offset]! ; direc. pre-indexado
4 op{L}{cond} coproc, CRd, [Rn], #{-}offset ; direc. post-indexado
5 op{L}{cond} coproc, CRd, label
6 op{L}{cond} coproc, CRd, [Rn], {option}
```

op	LDC / LDC2
cond	código condicional (En LDC2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
CRd	Registro del coprocesador, destino de la carga.
Rn	Registro core de ARM que contiene la dirección de memoria.
offset	Expresión signada modulo 4 (- es el signo) (0 a 1020).
!	Operador opcional que si se incluye, Rn e recarga con el offset.
label	Valor módulo 4 que representa un desplazamiento relativo al PC
opcion	opción para el coprocesador (0 a 255).

STC{2}Transfiere del Coprocesador a Memoria

Sintaxis:

```
1 op{L}{cond} coproc, CRd, [Rn]
2 op{L}{cond} coproc, CRd, [Rn, #{-}offset] ; direc. por offset
3 op{L}{cond} coproc, CRd, [Rn, #{-}offset]! ; direc. pre-indexado
4 op{L}{cond} coproc, CRd, [Rn], #{-}offset ; direc. post-indexado
5 op{L}{cond} coproc, CRd, [Rn], {option}
```

op	STC / STC2
cond	código condicional (En STC2 no aplica)
coproc	Nombre del coprocesador (p0-p15).
CRd	Registro del coprocesador, destino de la carga.
Rn	Registro core de ARM que contiene la dirección de memoria.
offset	Expresión signada modulo 4 (- es el signo) (0 a 1020).
!	Operador opcional que si se incluye, Rn e recarga con el offset.
opcion	opción para el coprocesador (0 a 255).

Temario

“Because embedded operating systems are designed for a specific purpose, historically embedded operating systems were simple, time constrained, and operated in limited memory. This distinction has changed over time as the sophistication of embedded hardware has increased. Features, traditionally found on desktop computers, such as virtual memory, have migrated into the embedded system world”.
ARM System Developers Guide.
Andrew N. Sloss, Dominic Symes, Chris Wright

El modelo Loop de control

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.
- Este ciclo se repetía infinitamente.

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.
- Este ciclo se repetía infinitamente.
- Los equipos trabajaban conectados a la red eléctrica.

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.
- Este ciclo se repetía infinitamente.
- Los equipos trabajaban conectados a la red eléctrica.
- ¿y el consumo de energía?, ¿baterías? ¿Movilidad?

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.
- Este ciclo se repetía infinitamente.
- Los equipos trabajaban conectados a la red eléctrica.
- ¿y el consumo de energía?, ¿baterías? ¿Movilidad?
- ¿Que es eso?

El modelo Loop de control

- Años 70. Primeros microcontroladores. Recursos mínimos. Complejidad mínima.
- Sistemas muy simples.
- Propósito: Controlar algunas pocas variables de entrada (sensores típicamente).
- En función de los resultados activar algún relé, o encender algún led. No mucho mas.
- Este ciclo se repetía infinitamente.
- Los equipos trabajaban conectados a la red eléctrica.
- ¿y el consumo de energía?, ¿baterías? ¿Movilidad?
- ¿Que es eso?
- ¿Un sistema Operativo? ¿Para que?

El modelo Loop de control

Programa típico escrito en lenguaje C y tenía éste aspecto:

El modelo Loop de control

Programa típico escrito en lenguaje C y tenía éste aspecto:

```
1  main()
2  {
3     init();
4     while(1)
5     {
6        Inputs_sence();
7        Values_process();
8        Output_control();
9     }
10 }
```

El modelo Loop de control

Programa típico escrito en lenguaje C y tenía éste aspecto:

```
1  main()
2  {
3     init();
4     while(1)
5     {
6        Inputs_sence();
7        Values_process();
8        Output_control();
9     }
10 }
```

Funciones sencillas. Hardware sencillo.

El modelo Loop de control

Programa típico escrito en lenguaje C y tenía éste aspecto:

```
1  main()
2  {
3     init();
4     while(1)
5     {
6        Inputs_sence();
7        Values_process();
8        Output_control();
9     }
10 }
```

Funciones sencillas. Hardware sencillo.

Estos sistemas de computo super simples eran un componente menor dentro de un sistema electrónico mucho mas complejo, en ocasiones con componentes mecánicos. Por eso se los empezó a denominar Embedded Systems (por estar “Integrados”dentro de un sistema mas amplio.).

Problemas del Modelo Loop

Problemas del Modelo Loop

- No sabría por donde empezar...

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.
- A medida que se fue sofisticando el nivel de requerimientos, ésta actividad de sensado perpetuo empezó a significar un consumo excesivo (y absolutamente pueril) de CPU.

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.
- A medida que se fue sofisticando el nivel de requerimientos, ésta actividad de sensado perpetuo empezó a significar un consumo excesivo (y absolutamente pueril) de CPU.
- Esto empezó a levantar temperatura y aumentar el consumo de los sistemas cuantas mas cosas sensaban

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.
- A medida que se fue sofisticando el nivel de requerimientos, ésta actividad de sensado perpetuo empezó a significar un consumo excesivo (y absolutamente pueril) de CPU.
- Esto empezó a levantar temperatura y aumentar el consumo de los sistemas cuantas mas cosas sensaban
- Podía pensarse en un esquema del tipo:

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.
- A medida que se fue sofisticando el nivel de requerimientos, ésta actividad de sensado perpetuo empezó a significar un consumo excesivo (y absolutamente pueril) de CPU.
- Esto empezó a levantar temperatura y aumentar el consumo de los sistemas cuantas mas cosas sensaban
- Podía pensarse en un esquema del tipo:

```
1    while (device_no_input)
```

Problemas del Modelo Loop

- No sabría por donde empezar...
- Cada paso es un Problema
- La CPU esta todo el tiempo trabajando en el sensado de los dispositivos de entrada.
- A medida que se fue sofisticando el nivel de requerimientos, ésta actividad de sensado perpetuo empezó a significar un consumo excesivo (y absolutamente pueril) de CPU.
- Esto empezó a levantar temperatura y aumentar el consumo de los sistemas cuantas mas cosas sensaban
- Podía pensarse en un esquema del tipo:

```
1    while (device_no_input)
```

- Pero si tenemos varios dispositivos bloqueo en uno que no recibe nada y pierdo lo que llegan por los demás.

Temario

Evento

Evento

En un Sistema un evento es una ocurrencia en la entrada que genera luego de un tiempo un efecto en la salida.
La propuesta es capturar ese evento y procesarlo mediante un bloque que maneje lo necesario dentro del sistema para que la respuesta sea la adecuada.
A este tipo de sistemas se los definió oportunamente como ***event-driven***, o ***event-reactive***.

Evento

En un Sistema un evento es una ocurrencia en la entrada que genera luego de un tiempo un efecto en la salida.
La propuesta es capturar ese evento y procesarlo mediante un bloque que maneje lo necesario dentro del sistema para que la respuesta sea la adecuada.
A este tipo de sistemas se los definió oportunamente como ***event-driven***, o ***event-reactive***.

Sincrónicos Eventos cuya ocurrencia es determinista, y periódica (Ej: timers).

Evento

En un Sistema un evento es una ocurrencia en la entrada que genera luego de un tiempo un efecto en la salida.
La propuesta es capturar ese evento y procesarlo mediante un bloque que maneje lo necesario dentro del sistema para que la respuesta sea la adecuada.
A este tipo de sistemas se los definió oportunamente como ***event-driven***, o ***event-reactive***.

Sincrónicos Eventos cuya ocurrencia es determinista, y periódica (Ej: timers).

Asincrónicos Eventos cuya ocurrencia no es determinista. Típicamente acciones de usuario, Por Ej: un click de mouse, pulsar una tecla, o relacionados con recepción de datos por una UART, I^2C, Ethernet, etc.

Sistemas Interrupt-driven

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).
- La Clave en este modelo en usar la CPU al mínimo cuando no se está procesando ningún evento.

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).
- La Clave en este modelo en usar la CPU al mínimo cuando no se está procesando ningún evento.
- La CPU debe estar lo mas inactiva que resulte posible.

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).
- La Clave en este modelo en usar la CPU al mínimo cuando no se está procesando ningún evento.
- La CPU debe estar lo mas inactiva que resulte posible.
- Así consumirá el mínimo de energía eléctrica como para mantener los datos en sus registros, y sus caches, al menos.

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).
- La Clave en este modelo en usar la CPU al mínimo cuando no se está procesando ningún evento.
- La CPU debe estar lo mas inactiva que resulte posible.
- Así consumirá el mínimo de energía eléctrica como para mantener los datos en sus registros, y sus caches, al menos.
- Desde el Cortex-A5 ARM dispone de una interrupción para poner al procesador en ese estado: **WFI** (**W**ait **F**or **I**nterrupt).

Sistemas Interrupt-driven

- Finalmente se terminan denominando de este modo en virtud de la asociación inequívoca entre evento e interrupción.
- En este caso el sistema está normalmente en un estado inactivo esperando que se produzca un evento (o varios tal vez).
- La Clave en este modelo en usar la CPU al mínimo cuando no se está procesando ningún evento.
- La CPU debe estar lo mas inactiva que resulte posible.
- Así consumirá el mínimo de energía eléctrica como para mantener los datos en sus registros, y sus caches, al menos.
- Desde el Cortex-A5 ARM dispone de una interrupción para poner al procesador en ese estado: **WFI** (**W**ait **F**or **I**nterrupt).
- En los procesadores ARM9 y ARM11 había que ir al Coprocesador cp15 para poner la CPU en este estado.

```
1     MOV r0, #0
2     MCR p15,0,R0,c7,c0,4    // Pone ARM9 en WFI state
```

Eventos sincrónicos

Eventos sincrónicos

- Usualmente depende de temporizadores.

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.
- En tal caso debe tenerse en cuenta que los procesadores ARM no soportan re entrancia en sus interrupciones.

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.
- En tal caso debe tenerse en cuenta que los procesadores ARM no soportan re entrancia en sus interrupciones.
- Una opción es quitar la tarea de la órbita del temporizador.

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.
- En tal caso debe tenerse en cuenta que los procesadores ARM no soportan re entrancia en sus interrupciones.
- Una opción es quitar la tarea de la órbita del temporizador.
- La otra es dividir su parte crítica para realizar dentro del ciclo de timer y lanzar una segunda en segundo plano que pueda realizarse sin urgencia.

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.
- En tal caso debe tenerse en cuenta que los procesadores ARM no soportan re entrancia en sus interrupciones.
- Una opción es quitar la tarea de la órbita del temporizador.
- La otra es dividir su parte crítica para realizar dentro del ciclo de timer y lanzar una segunda en segundo plano que pueda realizarse sin urgencia.
- Un ejemplo de este abordaje lo tenemos en Linux y se conoce como *Top Half* (parte crítica se ejecuta en todos los ciclos de timer) y *Bottom Half* (parte no crítica que puede pos datarse).

Eventos sincrónicos

- Usualmente depende de temporizadores.
- Es necesario que la tarea que debe manejar el evento no sea demasiado extensa, de lo contrario no se podría realizar en un intervalo completo de timer.
- En tal caso debe tenerse en cuenta que los procesadores ARM no soportan re entrancia en sus interrupciones.
- Una opción es quitar la tarea de la órbita del temporizador.
- La otra es dividir su parte crítica para realizar dentro del ciclo de timer y lanzar una segunda en segundo plano que pueda realizarse sin urgencia.
- Un ejemplo de este abordaje lo tenemos en Linux y se conoce como *Top Half* (parte crítica se ejecuta en todos los ciclos de timer) y *Bottom Half* (parte no crítica que puede pos datarse).
- No siempre es factible este particionado.

Eventos asincrónicos

Eventos asincrónicos

- No periódicos por naturaleza.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.
- Normalmente utilizan variables como flags para señalar la ocurrencia del/los evento/s.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.
- Normalmente utilizan variables como flags para señalar la ocurrencia del/los evento/s.
- Cuando se genera un evento se genera una interrupción.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.
- Normalmente utilizan variables como flags para señalar la ocurrencia del/los evento/s.
- Cuando se genera un evento se genera una interrupción.
- En el handler de la interrupción se leen los datos que eventualmente entregue el dispositivo asociado al evento, y se activa el flag asociado al evento.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.
- Normalmente utilizan variables como flags para señalar la ocurrencia del/los evento/s.
- Cuando se genera un evento se genera una interrupción.
- En el handler de la interrupción se leen los datos que eventualmente entregue el dispositivo asociado al evento, y se activa el flag asociado al evento.
- Cuando se procesa el evento se limpia el flag.

Eventos asincrónicos

- No periódicos por naturaleza.
- Ocurren aleatoriamente y en ocasiones tiene restricciones de tiempo de atención.
- Normalmente utilizan variables como flags para señalar la ocurrencia del/los evento/s.
- Cuando se genera un evento se genera una interrupción.
- En el handler de la interrupción se leen los datos que eventualmente entregue el dispositivo asociado al evento, y se activa el flag asociado al evento.
- Cuando se procesa el evento se limpia el flag.
- Es conveniente que al limpiar el flag se desactiven temporalmente las interrupciones para evitar carreras criticas (race conditions)

Prioridades de los eventos

Prioridades de los eventos

- En un sistema que maneja múltiples eventos es natural que éstos tengas diferentes prioridades.

Prioridades de los eventos

- En un sistema que maneja múltiples eventos es natural que éstos tengas diferentes prioridades.
- En este tipo de sistemas es conveniente un controlador de interrupciones que maneje adecuadamente las prioridades. De este modo se establece un primer nivel de gestión de prioridades.

Prioridades de los eventos

- En un sistema que maneja múltiples eventos es natural que éstos tengas diferentes prioridades.
- En este tipo de sistemas es conveniente un controlador de interrupciones que maneje adecuadamente las prioridades. De este modo se establece un primer nivel de gestión de prioridades.
- El orden de manejo de los eventos en el programa debe ser consistente con las prioridades del controlador de interrupciones.

Prioridades de los eventos

- En un sistema que maneja múltiples eventos es natural que éstos tengas diferentes prioridades.
- En este tipo de sistemas es conveniente un controlador de interrupciones que maneje adecuadamente las prioridades. De este modo se establece un primer nivel de gestión de prioridades.
- El orden de manejo de los eventos en el programa debe ser consistente con las prioridades del controlador de interrupciones.
- Los handlers de eventos pueden ser implementados por unidades de código independientes en forma de procesos o tareas, que a su vez pueden ser despachados para ejecución en un determinado orden de prioridades.

Estructura de un modelo orientado a eventos

```
1  int do_event1()
2  {
3      respuesta1_urgente();
4      event1 = 1;
5  }
6  int do_event2()
7  {
8      respuesta2_urgente();
9      event2 = 1;
10 }
11 int main()
12 {
13     init(); // Software de inicializacion
14     while(1)
15     {      // main program: check event flags; handle events
16         if (event1)
17         {
18             completa_respuesta1();
19             lock(); event1 = 0; unlock(); // event1 a 0
20         }
21         if (event2)
22         {
23             completa_respuesta2();
24             lock(); event2 = 0; unlock(); // event2 a 0
25         }
26         asm("WFI"); // power saving mode
27     }
28 }
```

Deficiencias del manejo en loop

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.
- El handler de interrupción debe ser muy corto a fin de evitar reentrancias. Esto deriva en que parte de la tarea se resuelve en el lazo principal.

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.
- El handler de interrupción debe ser muy corto a fin de evitar reentrancias. Esto deriva en que parte de la tarea se resuelve en el lazo principal.
- Incluso en los eventos periódicos se necesita trabajar fuera del handler del timer para no perder ticks.

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.
- El handler de interrupción debe ser muy corto a fin de evitar reentrancias. Esto deriva en que parte de la tarea se resuelve en el lazo principal.
- Incluso en los eventos periódicos se necesita trabajar fuera del handler del timer para no perder ticks.
- En cada interrupción es necesario preguntar por todos los flags de los posibles eventos. Sería deseable que por cada evento se active solamente su handler, y el resto ni se encueste.

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.
- El handler de interrupción debe ser muy corto a fin de evitar reentrancias. Esto deriva en que parte de la tarea se resuelve en el lazo principal.
- Incluso en los eventos periódicos se necesita trabajar fuera del handler del timer para no perder ticks.
- En cada interrupción es necesario preguntar por todos los flags de los posibles eventos. Sería deseable que por cada evento se active solamente su handler, y el resto ni se encueste.
- Los eventos pueden extenderse mas allá de interrupciones y entradas de usuario y responder a sincronización o comunicaciones.

Deficiencias del manejo en loop

- Aun cuando manejen estado de power-saving, los lazos en el main presentan algunas deficiencias.
- El handler de interrupción debe ser muy corto a fin de evitar reentrancias. Esto deriva en que parte de la tarea se resuelve en el lazo principal.
- Incluso en los eventos periódicos se necesita trabajar fuera del handler del timer para no perder ticks.
- En cada interrupción es necesario preguntar por todos los flags de los posibles eventos. Sería deseable que por cada evento se active solamente su handler, y el resto ni se encueste.
- Los eventos pueden extenderse mas allá de interrupciones y entradas de usuario y responder a sincronización o comunicaciones.
- Para superar estos inconvenientes es necesario un modelo de procesos.

Temario

Conceptos vistos

Conceptos vistos

Un sistema se conforma con múltiples procesos que ejecutan en forma concurrente.
Un proceso es una entidad de ejecución (código datos y pila) que puede ser despachado para su ejecución, detenido para cederle la CPU a otro proceso, y reasumido tiempo después a partir del punto exacto en el que fue suspendido.

Modelos de Proceso

Modelos de Proceso

Uniprocesador Tiene una sola CPU. Los procesos ejecutan concurrentemente mediante un modelo multitasking

Modelos de Proceso

Uniprocesador Tiene una sola CPU. Los procesos ejecutan concurrentemente mediante un modelo multitasking

Multiprocesador Los sistemas MP tienen mas de una CPU. Las CPUs pueden ser idénticas (Simétricos, SMP) o diversas (Asimétricos AMP). Los procesos ejecutan paralelamente en cada una de las CPUs del sistema, las cuales, sin perjuicio de ésto último, a su vez pueden implementar multitasking para ejecutar procesos en forma concurrente.

Modelos de Proceso

Uniprocesador Tiene una sola CPU. Los procesos ejecutan concurrentemente mediante un modelo multitasking

Multiprocesador Los sistemas MP tienen mas de una CPU. Las CPUs pueden ser idénticas (Simétricos, SMP) o diversas (Asimétricos AMP). Los procesos ejecutan paralelamente en cada una de las CPUs del sistema, las cuales, sin perjuicio de ésto último, a su vez pueden implementar multitasking para ejecutar procesos en forma concurrente.

Real Address Space El sistema no tiene o no utiliza MMU por restricciones de respuesta temporal. No se provee mapeo de memoria. Los procesos ejecutan en el mismo espacio de memoria. No hay protección de memoria por hardware

Modelos de Proceso

Modelos de Proceso

Virtual Address Space

El sistema utiliza MMU proveyendo mapeo de memoria. Los procesos se ejecutan en modo kernel o en modo user. Cuando ejecutan en modo kernel comparten el espacio de memoria del kernel. Cuando ejecutan en modo user el mapeo de memoria le asegura a cada proceso su propio espacio protegido de memoria.

Modelos de Proceso

Virtual Address Space El sistema utiliza MMU proveyendo mapeo de memoria. Los procesos se ejecutan en modo kernel o en modo user. Cuando ejecutan en modo kernel comparten el espacio de memoria del kernel. Cuando ejecutan en modo user el mapeo de memoria le asegura a cada proceso su propio espacio protegido de memoria.

Modelo Estático Todos los procesos se crean en el momento en que arranca el sistema y permanecen en memoria permanentemente. Cada proceso puede ser periódico o event-driven. El scheduling puede ser por prioridad, o sin anticipación (non preemptive), es decir, los procesos ejecutan hasta que liberan la CPU por si mismos.

Modelos de Proceso

Modelos de Proceso

Modelo Dinámico Los procesos se crean en forma dinámica ya sea como respuesta a eventos, en forma periódica, o como respuesta a una acción o comando del usuario. Cuando un proceso finaliza su ejecución es removido de memoria y se liberan todos los recursos que se le habían asignado.

Modelos de Proceso

Modelo Dinámico Los procesos se crean en forma dinámica ya sea como respuesta a eventos, en forma periódica, o como respuesta a una acción o comando del usuario. Cuando un proceso finaliza su ejecución es removido de memoria y se liberan todos los recursos que se le habían asignado.

Non-preemptive Los procesos ejecutan hasta que ceden la CPU a otro proceso o hasta que entran en un estado de espera de un evento. Nunca son suspendidos en forma forzada para pasar el control a otro proceso.

Modelos de Proceso

Modelo Dinámico Los procesos se crean en forma dinámica ya sea como respuesta a eventos, en forma periódica, o como respuesta a una acción o comando del usuario. Cuando un proceso finaliza su ejecución es removido de memoria y se liberan todos los recursos que se le habían asignado.

Non-preemptive Los procesos ejecutan hasta que ceden la CPU a otro proceso o hasta que entran en un estado de espera de un evento. Nunca son suspendidos en forma forzada para pasar el control a otro proceso.

Preemptive La CPU puede ser asignada a un determinado proceso en cualquier momento quitándosela a otro proceso, por políticas de prioridad de cualquier otro tipo.

Temario

Clasificación

Clasificación

La clasificación de Modelos de proceso anterior no es excluyente. Quiere decir que un sistema embedded de acuerdo con sus requerimientos puede incluir en su diseño una combinación de los Modelos de Proceso anteriores.

Clasificación

La clasificación de Modelos de proceso anterior no es excluyente. Quiere decir que un sistema embedded de acuerdo con sus requerimientos puede incluir en su diseño una combinación de los Modelos de Proceso anteriores.

- Kernel Monoprocesador.

Clasificación

La clasificación de Modelos de proceso anterior no es excluyente. Quiere decir que un sistema embedded de acuerdo con sus requerimientos puede incluir en su diseño una combinación de los Modelos de Proceso anteriores.

- Kernel Monoprocesador.
- Sistema Operativo Monoprocesador.

Clasificación

La clasificación de Modelos de proceso anterior no es excluyente. Quiere decir que un sistema embedded de acuerdo con sus requerimientos puede incluir en su diseño una combinación de los Modelos de Proceso anteriores.

- Kernel Monoprocesador.
- Sistema Operativo Monoprocesador.
- Sistema Multiprocesador.

Clasificación

La clasificación de Modelos de proceso anterior no es excluyente. Quiere decir que un sistema embedded de acuerdo con sus requerimientos puede incluir en su diseño una combinación de los Modelos de Proceso anteriores.

- Kernel Monoprocesador.
- Sistema Operativo Monoprocesador.
- Sistema Multiprocesador.
- Sistema Real Time.

Kernel Monoprocesador

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.
- Los procesos ejecutan en la misma imagen de memoria del Kernel.

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.
- Los procesos ejecutan en la misma imagen de memoria del Kernel.
- Los procesos pueden ser estáticos o dinámicos.

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.
- Los procesos ejecutan en la misma imagen de memoria del Kernel.
- Los procesos pueden ser estáticos o dinámicos.
- El esquema de prioridad es estático.

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.
- Los procesos ejecutan en la misma imagen de memoria del Kernel.
- Los procesos pueden ser estáticos o dinámicos.
- El esquema de prioridad es estático.
- Se comporta como un sistema operativo Non Preemptive.

Kernel Monoprocesador

- Una sola CPU, sin Hardware para manejo de memoria.
- Los procesos ejecutan en la misma imagen de memoria del Kernel.
- Los procesos pueden ser estáticos o dinámicos.
- El esquema de prioridad es estático.
- Se comporta como un sistema operativo Non Preemptive.
- Se aplica a microcontroladores simples.

Sistema Operativo Monoprocesador

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.
- Los procesos pueden compartir objetos únicamente en el espacio del Kernel.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.
- Los procesos pueden compartir objetos únicamente en el espacio del Kernel.
- En modo kernel un proceso ejecuta hasta que cede el control voluntariamente, es decir es un esquema Non Preemptive.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.
- Los procesos pueden compartir objetos únicamente en el espacio del Kernel.
- En modo kernel un proceso ejecuta hasta que cede el control voluntariamente, es decir es un esquema Non Preemptive.
- En modo User cualquier proceso puede ser anticipado (preempted) por otro de mayor prioridad al que le cederá el control.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.
- Los procesos pueden compartir objetos únicamente en el espacio del Kernel.
- En modo kernel un proceso ejecuta hasta que cede el control voluntariamente, es decir es un esquema Non Preemptive.
- En modo User cualquier proceso puede ser anticipado (preempted) por otro de mayor prioridad al que le cederá el control.
- Los procesos pueden ser estáticos o dinámicos.

Sistema Operativo Monoprocesador

- Una sola CPU, con una MMU que provee facilidades de *memory mapping* generalmente en base al método de Paginación.
- En modo User los procesos ejecutan protegidos dentro de su propio espacio de direccionamiento provisto por el Kernel.
- En Modo Kernel los procesos ejecutan en el espacio de direccionamiento del Kernel.
- Los procesos pueden compartir objetos únicamente en el espacio del Kernel.
- En modo kernel un proceso ejecuta hasta que cede el control voluntariamente, es decir es un esquema Non Preemptive.
- En modo User cualquier proceso puede ser anticipado (preempted) por otro de mayor prioridad al que le cederá el control.
- Los procesos pueden ser estáticos o dinámicos.
- Se aplica Microprocesadores de propósito general.

Sistema Multiprocesador

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.
- Todas las CPUs comparten el mismo espacio de memoria.

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.
- Todas las CPUs comparten el mismo espacio de memoria.
- Los procesos ejecutan en CPUs diferentes en paralelo.

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.
- Todas las CPUs comparten el mismo espacio de memoria.
- Los procesos ejecutan en CPUs diferentes en paralelo.
- Se requieren técnicas de programación concurrente.

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.
- Todas las CPUs comparten el mismo espacio de memoria.
- Los procesos ejecutan en CPUs diferentes en paralelo.
- Se requieren técnicas de programación concurrente.
- Se requieren técnicas de sincronización y protección mas estrictas y sofisticadas.

Sistema Multiprocesador

- El sistema tiene múltiples CPUs, en el mismo core o en diferentes cores.
- Todas las CPUs comparten el mismo espacio de memoria.
- Los procesos ejecutan en CPUs diferentes en paralelo.
- Se requieren técnicas de programación concurrente.
- Se requieren técnicas de sincronización y protección mas estrictas y sofisticadas.
- Se aplica a sistemas en un rango de sistemas muy amplio.

Sistemas Real Time (RT)

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.
- Las interrupciones deben ser atendidas con mínima demora y resueltas en lapsos muy breves.

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.
- Las interrupciones deben ser atendidas con mínima demora y resueltas en lapsos muy breves.
- Estos requisitos pueden ser tomados son pautas, sin garantía que sean alcanzables.

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.
- Las interrupciones deben ser atendidas con mínima demora y resueltas en lapsos muy breves.
- Estos requisitos pueden ser tomados son pautas, sin garantía que sean alcanzables.
- En cambio si el sistema se diseña para Real Time, estas restricciones temporales se deben asumir muy rigurosamente.

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.
- Las interrupciones deben ser atendidas con mínima demora y resueltas en lapsos muy breves.
- Estos requisitos pueden ser tomados son pautas, sin garantía que sean alcanzables.
- En cambio si el sistema se diseña para Real Time, estas restricciones temporales se deben asumir muy rigurosamente.
- Se puede tener este abordaje en sistemas multi o mono procesador, o en microcontroladores sencillos.

Sistemas Real Time (RT)

- Los sistemas embebidos generalmente tienen requerimientos temporales específicos.
- Las interrupciones deben ser atendidas con mínima demora y resueltas en lapsos muy breves.
- Estos requisitos pueden ser tomados son pautas, sin garantía que sean alcanzables.
- En cambio si el sistema se diseña para Real Time, estas restricciones temporales se deben asumir muy rigurosamente.
- Se puede tener este abordaje en sistemas multi o mono procesador, o en microcontroladores sencillos.
- Los procesos pueden ser estáticos o dinámicos.