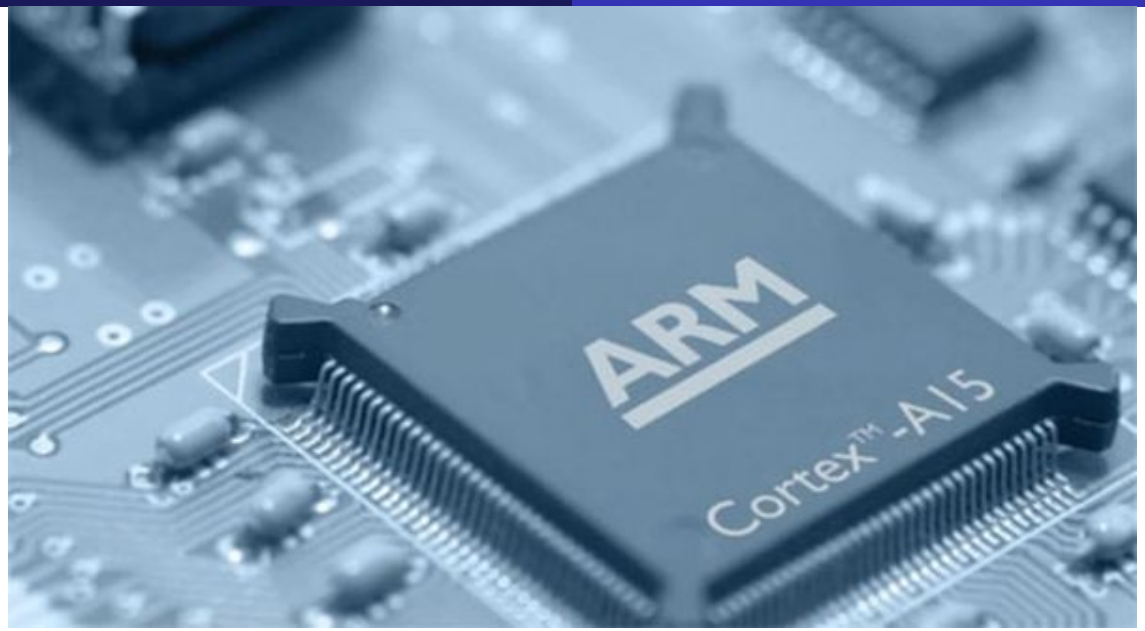


Procesadores ARM

Alejandro Furfaro

27 de abril de 2020

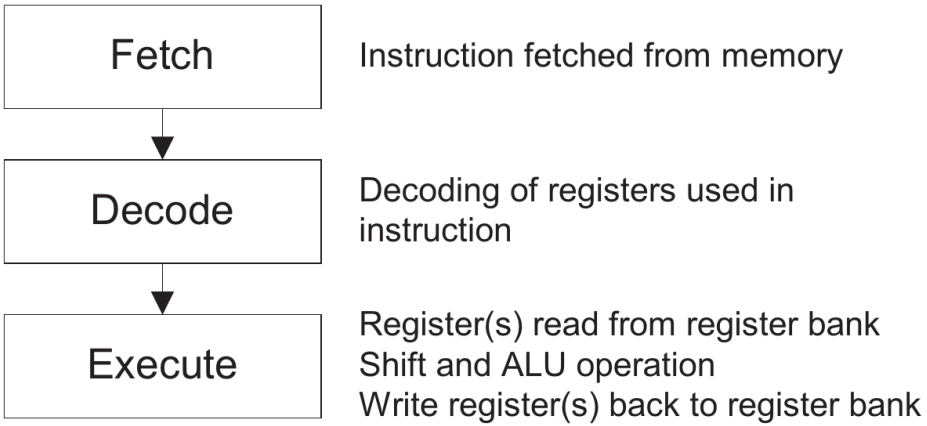
Agenda

Un Modelo de Negocio muy original

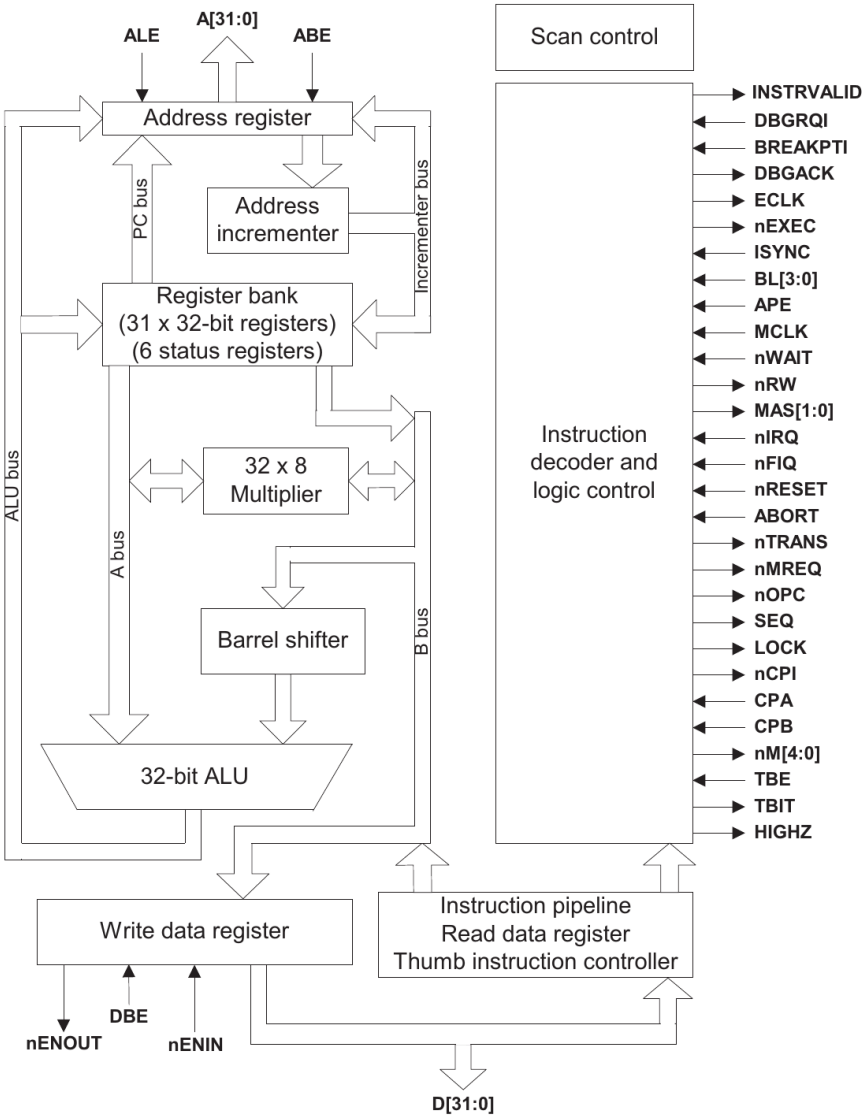
- ARM1: primer procesador creado por Acorn Computer en 1985. Apuntaba al mercado desktop (dominado por Intel y Motorola).
- Evoluciona en ARM2 y ARM3. Los únicos nichos eran educación, y hobbistas fundamentalmente.
- Aparecen empresas interesada en integrar procesadores ARM en sus propios diseños de sistemas embebidos. En especial Apple.
- Nace Advanced RISC Machine (ARM), producto del aporte de capital de Apple y know how e ingeniería de Acorn.
- Su modelo de negocios será diferente del clásico. *En lugar de vender el chip con el procesador, vende los derechos de propiedad intelectual para que otras compañías empleen procesadores ARM en sus diseños embebidos*.

1er. Hito ARM7TDMI

1. Fue el procesador elegido por Apple para el iPod.
2. ***D*** por **D**ebug Interface.
3. ***I*** por **I**CE o In Circuit Emulation
4. ***M*** por **M**ultiplier (incluido en el datapath).
5. Set de instrucciones Thumb
6. pipeline de tres etapas

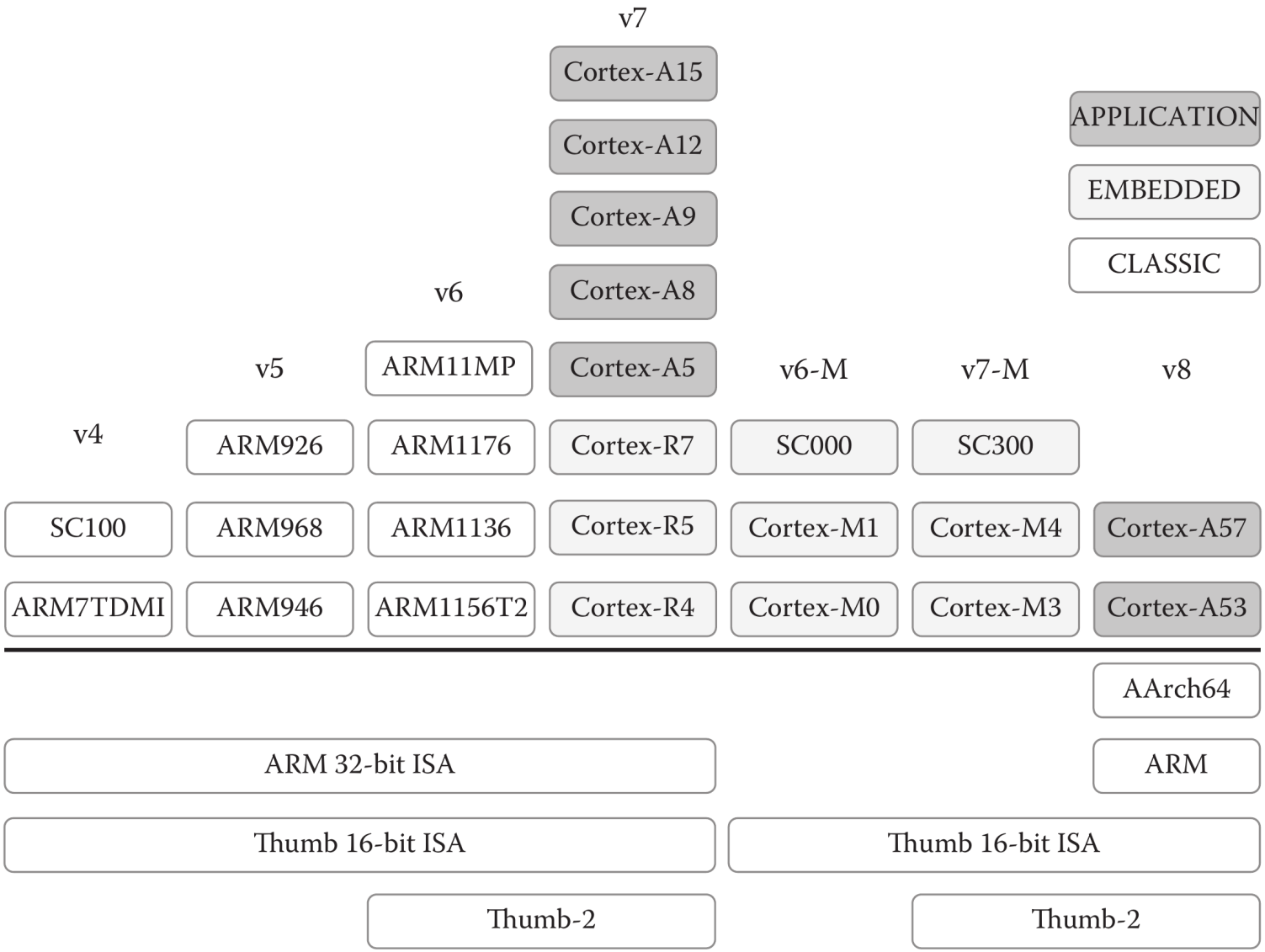


ARM7TDMI - Diagrama



- Claramente modelo Von Newmann
- 6 Modos de Operación.
- 31 registros. Pero en cada momento hay 16 visibles
- Incluye Multiplicación. . . No era RISC?
- El Modo Thumb fué altamente adoptado a medida que este procesador empezó a utilizarse en los teléfonos móviles para economizar memoria.

ARM - Genealogía



Evolución de versiones de Arquitectura

Límites cada vez mas difusos. . .

¿Cual es el rango de un Sistema Embebido?

- Una pregunta cada vez mas difícil de responder.
- ¿Es un controlador de velocidad de un motor basado en un sencillo micro controlador un SE?
- ¿Es un Navegador satelital basado en Linux para un automóvil un SE?
- ¿Es un teléfono móvil basado en un SO como Android que ejecuta sobre un hardware de varios cores un SE?

¿Cual es la respuesta correcta

- Todo indica que todas son correctas.
- Hace algunos años si alguien decía que un sistema embebido es uno basado en un micro controlador, tal vez estuviese en lo cierto.
- La marca comercial CORTEX, que describimos en el slide siguiente parece indicar que esta afirmación ya no es correcta.

CORTEX

ARM presenta la arquitectura ARMv7

Probablemente en la linea de razonamiento del slide anterior, al presentar ARMv7, ARM introduce la marca CORTEX a la que a su vez divide entre tres familias (o perfiles) diferentes de acuerdo con el tipo de Sistema embebido que se necesite desarrollar.

CORTEX-A

A de **A**pplication. Pensada para sistemas High-End. Smartphnes, Tables, Notebooks, y algún día tal vez servers

CORTEX-R

R por **R**eal Time. Es muy similar al **A** solo que su MMU no tiene capacidad de gestión dinámica de memoria.

CORTEX-M

M por **M**icrocontroller. Simplificación de la arquitectura v7, para adaptarla a Microcontroladores. Rompe la v7 en algún sentido.

Arquitectura Cortex-A y CORTEX-R

Objetivos de la ARMv7

De la lectura de las primeras páginas de “ARM Architecture Reference Manual, ARMv7-A and ARMv7-R Edition”, puede notarse que ARM se propone definir los aspectos arquitecturales de procesadores RISC que sean capaces de implementar un amplio rango de soluciones mediante sistemas de pequeño tamaño, lo cual produce un consumo los mas bajo posible, manteniendo niveles de performance aceptables. Si lo logra o no será materia de análisis luego de experimentar lo suficiente.

- Set de 16 Registros de Propósito General.
- Acceso a memoria Load / Store. (Los operandos de instrucciones aritmético lógicas son Registros).
- Los accesos a memoria son simples, mediante el uso de registros índices.

Arquitectura Cortex-A y Cortex-R

- Tiene instrucciones que combinan desplazamientos con operaciones aritméticas y lógicas.
- Instrucciones de auto incremento/decremento de direcciones para optimizar loops.
- Load/Store Múltiples para optimizar operaciones de acceso a memoria.
- Mecanismos para Ejecución Condicional de Instrucciones para acelerar ejecución (una perlita sin dudas).

Arquitectura Cortex-M

Objetivos de la ARMv7

De la lectura de las primeras páginas de “ARMv7-M Architecture Reference Manual”, puede notarse la diferencia.

- Capacidad de implementar soluciones de aplicación industrial líderes en bajo consumo de energía, performance, restricciones de área, que proporcionen además:
- Operación altamente determinística:
  - La mayoría de las operaciones ejecutan en un ciclo de clock o en el peor caso muy pocos ciclos.
  - Pipeline de pocas etapas de modo de introducir latencia mínima en atención de interrupciones.
  - Operación sin cache.

Arquitectura Cortex-M

Objetivos de la ARMv7

De la lectura de las primeras páginas de “ARMv7-M Architecture Reference Manual”, puede notarse la diferencia.

- Integración para ser programado en C/C++. Los manejadores de excepciones e interrupciones se escriben directamente como funciones C/C++, utilizando convención de llamadas standard.
- Baja cantidad de pines de interconexión para simplificar el diseño de hardware.
- Soporte para debug y software profiling para sistemas manejados por eventos (event driven systems).

Primeras Conclusiones

- Tal parece que la arquitectura real es la implementada en los perfiles A y R.
- La ARMv7-M, como veremos tiene algunas diferencias algo menores respecto de las A y R,
- Sin embargo parece ser mas que nada una modificación de organización que de arquitectura.
- Relean mas detenidamente el slide anterior donde se decribe el perfil v7-M.
- Ninguna de las características allí descriptas correponde a la arquitectura. No dependen del set de instrucciones ni de los registros.

Set de instrucciones

- El set de instrucciones ARM es de 32 bits, tamaño fijo, y campos de operandos tambien fijo, de acuerdo con los lineamientos RISC.
- Con el objetivo de lograr una mayor densidad de código ARM desarrollo un set de insrucciones de 16 bits de tamaño, y lo denominó ***Thumb***.
- El costo de esta mejora en densidad es la performance.
- Por otra parte, ***Thumb*** es un subset del set original de 32 bits (que se denomina ***arm***)
- En segmentos críticos de código como por ejemplo un handler de interrupción, es posible pasar temporariamente de ejecutar instrucciones ***thumb*** y ejecutar instrucciones de 32 bits (***arm***)

Set de instrucciones

- Luego de ARMv6, ARM presenta la arquitectura ARMv6T2, que introduce la Tecnología ***Thumb-2***.
- Agrega una serie de instrucciones de 32 bits que permiten aproximar en gran medida la performane con el modo ***arm*** puro, a la vez que mantiene una baja densidad de código.

Extensiones

- En general los diseñadores de Procesadores RISC al principio utilizan la menor cantidad posible de instrucciones.
- En la medida que se busca que este procesador sea mas competitivo dentro de su segmento, se agregan sub sets de instrucciones para enriquecer su capacidad de procesamiento.
- Normalmente estos sub sets se conocen como Extensiones de Arquitectura.
- ARM tiene varias. Algunas no han logrado despertar demasiado entusiasmo, otras han sido declaradas obsoletas como ***ThumbEE***. Otras son de aplicación creciente.
- Claro está. . . si se implementan todas las extensiones, los postulados RISC quedan bastante atrás. . .

Extensiones

- ***Jazzele***. Es un set de instrucciones capaces de ejecutar byte code Java, evitando el uso de Java Runtime Environment para procesar dicho código, con el propósito de acelerar la ejecución de Java. No vamos a ocuparnos de ella en el curso, pero para entender su estado, a partir de la ARMv6, se requiere una implementación al menos trivial de estas instrucciones. No obstante, se la sigue considerando una extensión.

- ***ThumbEE***. Thumb Excecution Environment, se pensó como una suerte de soporte a JIT (Just In Time) compiler, que permita generar el código Thumb justo antes de ejecutarse. Es decir permite ayudar en la ejecución de código generado dinámicamente, o para otros lenguajes que generen byte code como Dalvik / Android. Está oficialmente obsoleta desde el release C de la ARMv7 (2011).

Extensiones para Punto Flotante

- Es una prestación obligada para aplicaciones de cálculo avanzado.
- No es imprescindible en todos los sistemas. Por eso es razonable que sea una extensión.
- Aquí encontramos diferentes extensiones para ARMv7-A y ARMv7-R, respecto de ARMv7-M
- Para RAMv7-A y ARMv7-R las versiones VFPv3 y vFPv4 establecen los formatos de datos soportados, y los registros involucrados e instrucciones, que se introducen. Trabajan con Single y Half Precision
- Para ARMv7-M las versiones FPv4-SP y VFPv5 definen extensiones mínimas para esta arquitectura en relación a las prestaciones que se implementan en las versiones ARMv7-A y R.

Extensiones para DSP

- Es muy útil cuando se requiere implementar los algoritmos clásicos de Procesamiento Digital de Señales, para procesar audio, video, o imágenes.
- No es imprescindible en todos los sistemas. Por eso es razonable que sea una extensión.
- Nuevamente encontramos diferentes extensiones para ARMv7-A y ARMv7-R, respecto de ARMv7-M
- Para RAMv7-A y ARMv7-R estas extensiones están muy relacionadas con las versiones VFPv3 y vFPv4 de punto flotante vistas en el slide anterior.
- Para ARMv7-M se definen algunas instrucciones que derivan en una implementación mas mas modesta de esta extensión.

Extensiones mas avanzadas

- La arquitectura ARMv7 tiene para los perfiles A y R extensiones avanzadas.
- Security Extensions.
- Multiprocessor Extensions
- Large Physical Address Extensions. Requiere de Multiprocessor Extensions. Lleva a 40 bits la capacidad de direccionamiento.
- Virtualization Extensions. Requiere Security y Large Physical Address Extensions.
- Generic Timer Extensions.
- Performance Monitoring Extensions

Modelo de Memoria

Definiciones de ARMv7 respecto de la memoria

- Organizada como un espacio flat continuo de 2^{32} bytes. Puede considerarse también como un espacio de 2^{30} palabras de 32 bits o 2^{31} palabras de 16 bits.
- La CPU puede generar excepciones ante accesos a datos no alineados.
- Tiene capacidad de restringir el acceso de las aplicaciones a determinadas áreas de memoria.
- Traducción de dirección virtual a dirección física.
- Alterar la interpretación de un datos de 16 o 32 bits entre Little Endian o Big Endian.
- Controlar el orden de los accesos, y control de caches
- Sincronizar el acceso a memoria cuando se tiene mas de una CPU en el sistema.

Dos modelos de programación

Siempre que un procesador moderno implementa un modo de trabajo para las aplicaciones, y otro mas Privilegiado para el Sistema Operativo, o cualquier tipo de software de administración ad-hoc, hay dos modelos de programación diferentes, y entre ambos conforman la arquitectura:

- Modelo de Programación de Aplicaciones
  Se compone de los registros de arquitectura en instrucciones accesibles independientemente del modo de trabajo

- Modelo de Programación de Sistemas.
  Se compone de los registros e instrucciones del modelo de Programación de Aplicaciones, mas un set de registros e instrucciones que solo pueden ser accedidos por código que ejecuta en nivel privilegiado

Tipo y Tamaño de datos

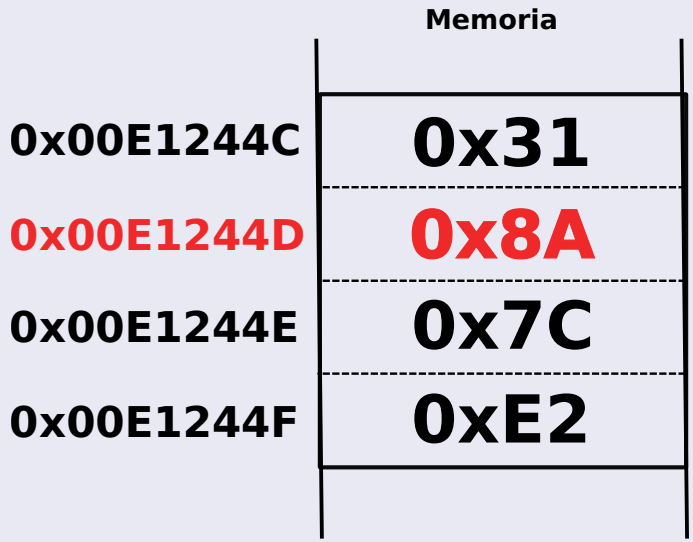
- El set de instrucciones soporta en memoria estos tipos de datos y sus tamaños:
  - Byte $\rightarrow$ 8 bits (1 byte)
  - Halfword $\rightarrow$ 16 bits (2 bytes)
  - Word $\rightarrow$ 32 bits (4 bytes)
  - Doubleword* $\rightarrow$ 64 bits (8 bytes)
- El set de instrucciones trabaja con los siguientes tipos de datos en los registros de arquitectura
  - Punteros de 32-bit.
  - Enteros Signados o No signados de 32-bit.
  - Enteros No Signados de 16-bit u 8-bit, en formato zero-extended.
  - Enteros Signados de 16-bit u 8-bit, en formato sign-extended.
  - Dos enteros de 16-bit empaquetados en un registro.*
  - Cuatro enteros de 8-bit empaquetados en un registro.*
  - Entero No Signado or Signado de 64-bit mantenido en dos registros de 32-bit.

* Solo ARMv7-A y ARMv7-R

Endianess

Layout de un byte en memoria

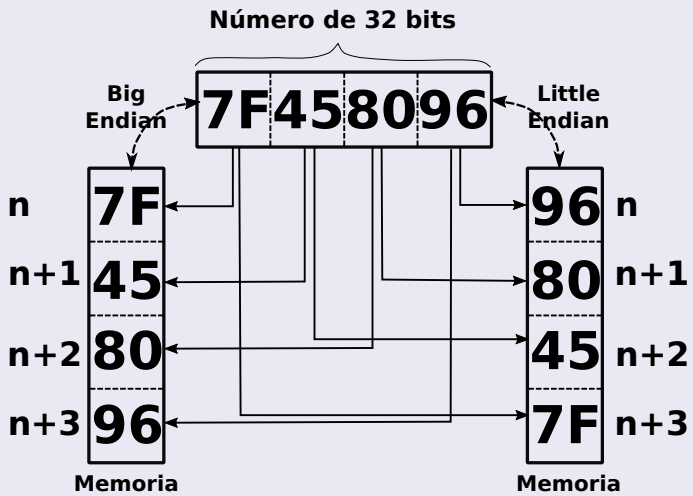
- Si queremos almacenar un byte no hay ambigüedades posibles.
- Ej: Almacenar el valor 0x8A en la dirección de memoria 0x00E1244D.



Endianess

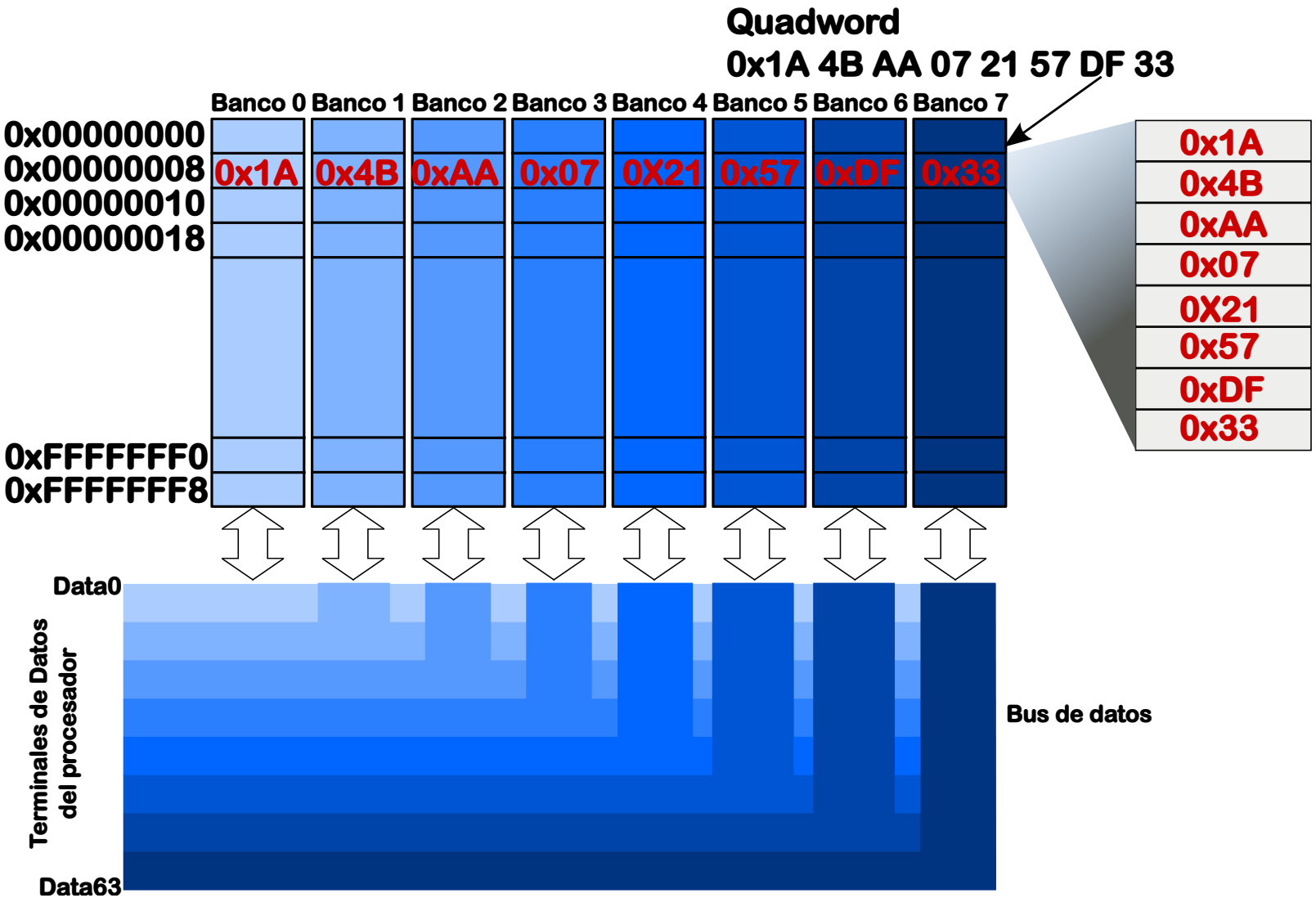
Layout de un half word, word o doble word en memoria

- Cuando representamos un valor de varios bytes el orden en el que se almacena no es trivial.
- ¿Como almacenás el número 0x7F458096 a partir de la dirección de memoria n?

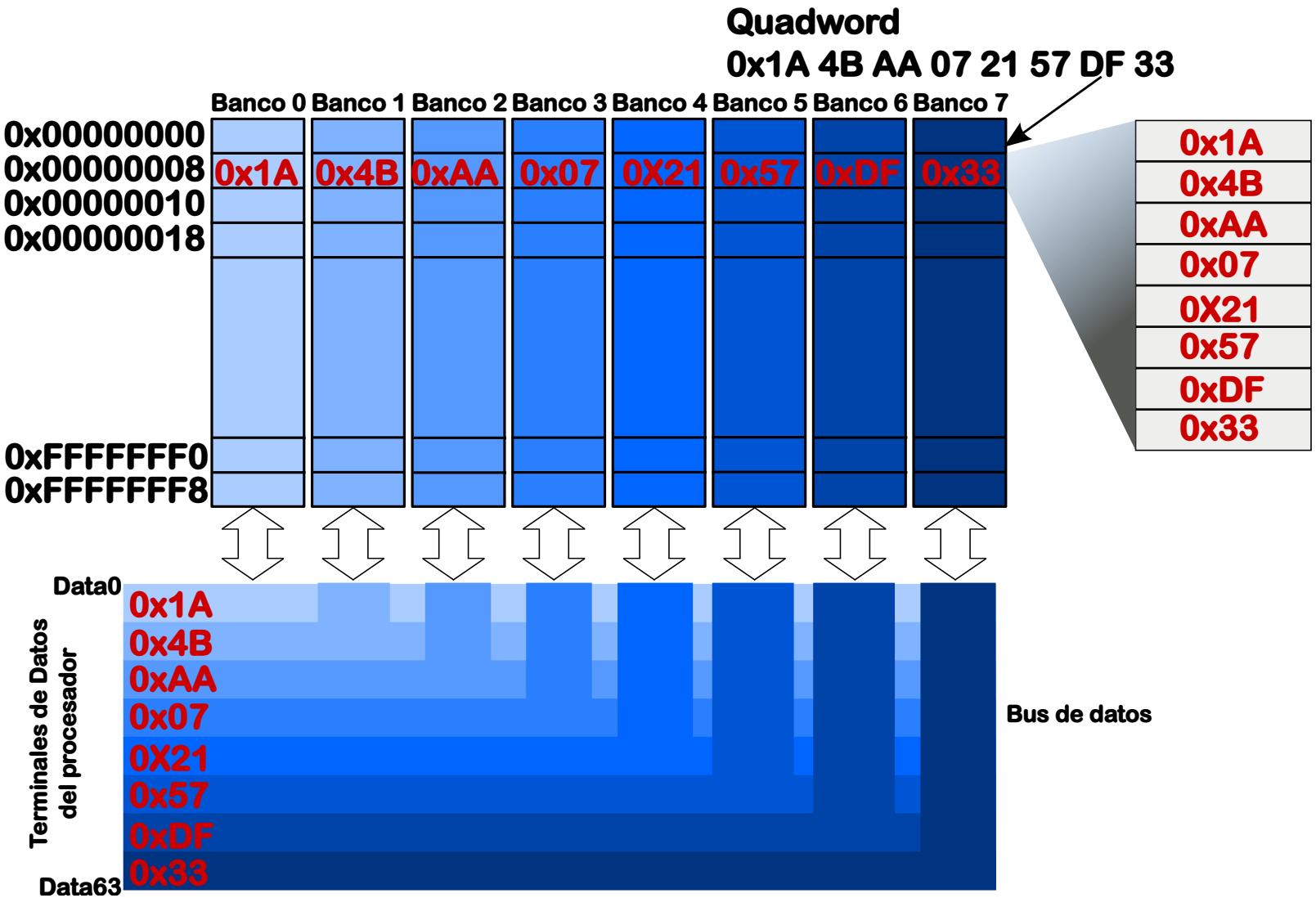


Los Procesadores CORTEX-A y CORTEX-R soportan Little Endian y Big Endian. CORTEX-M solo Little Endian

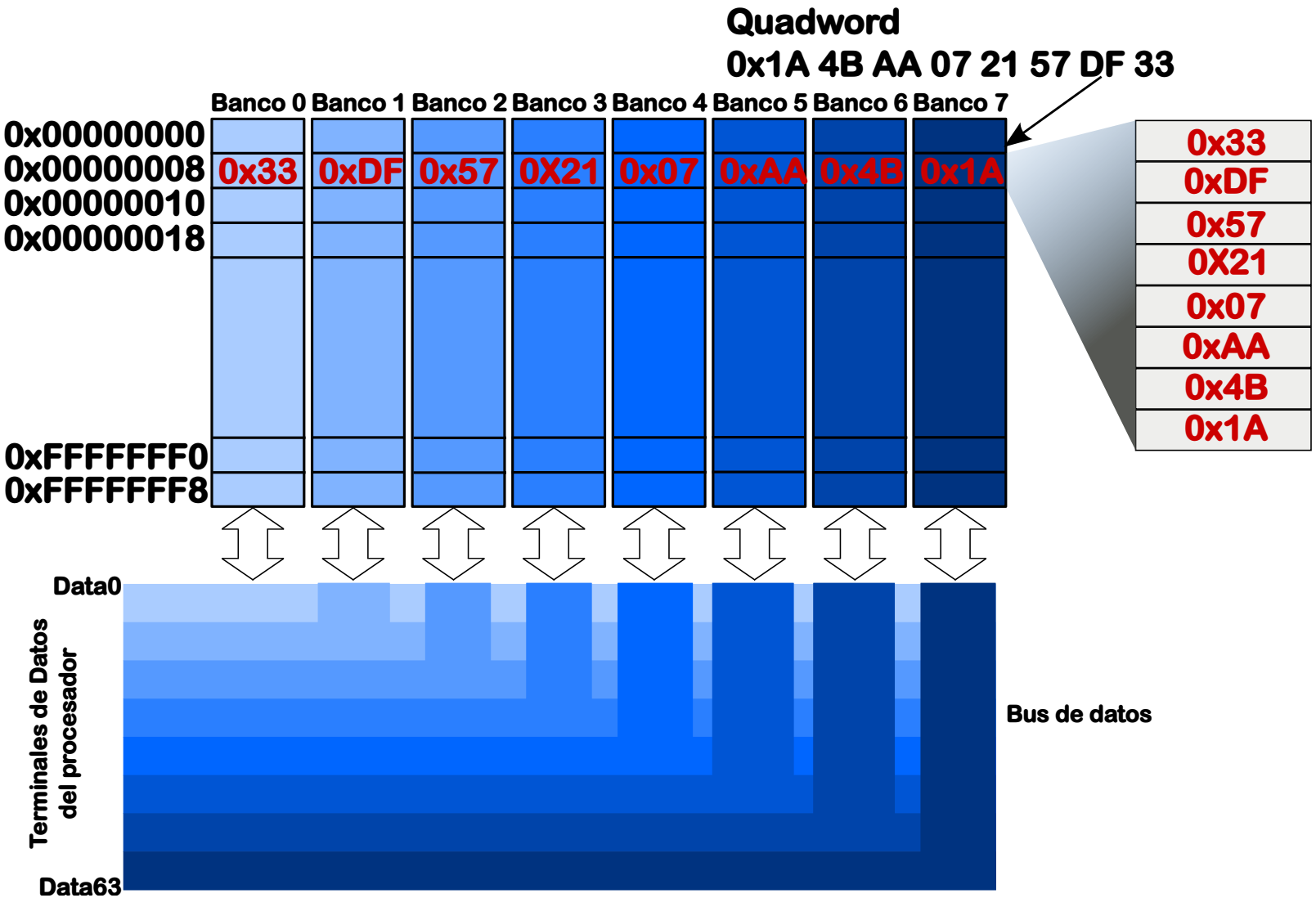
Aquí está el dato “al derecho”



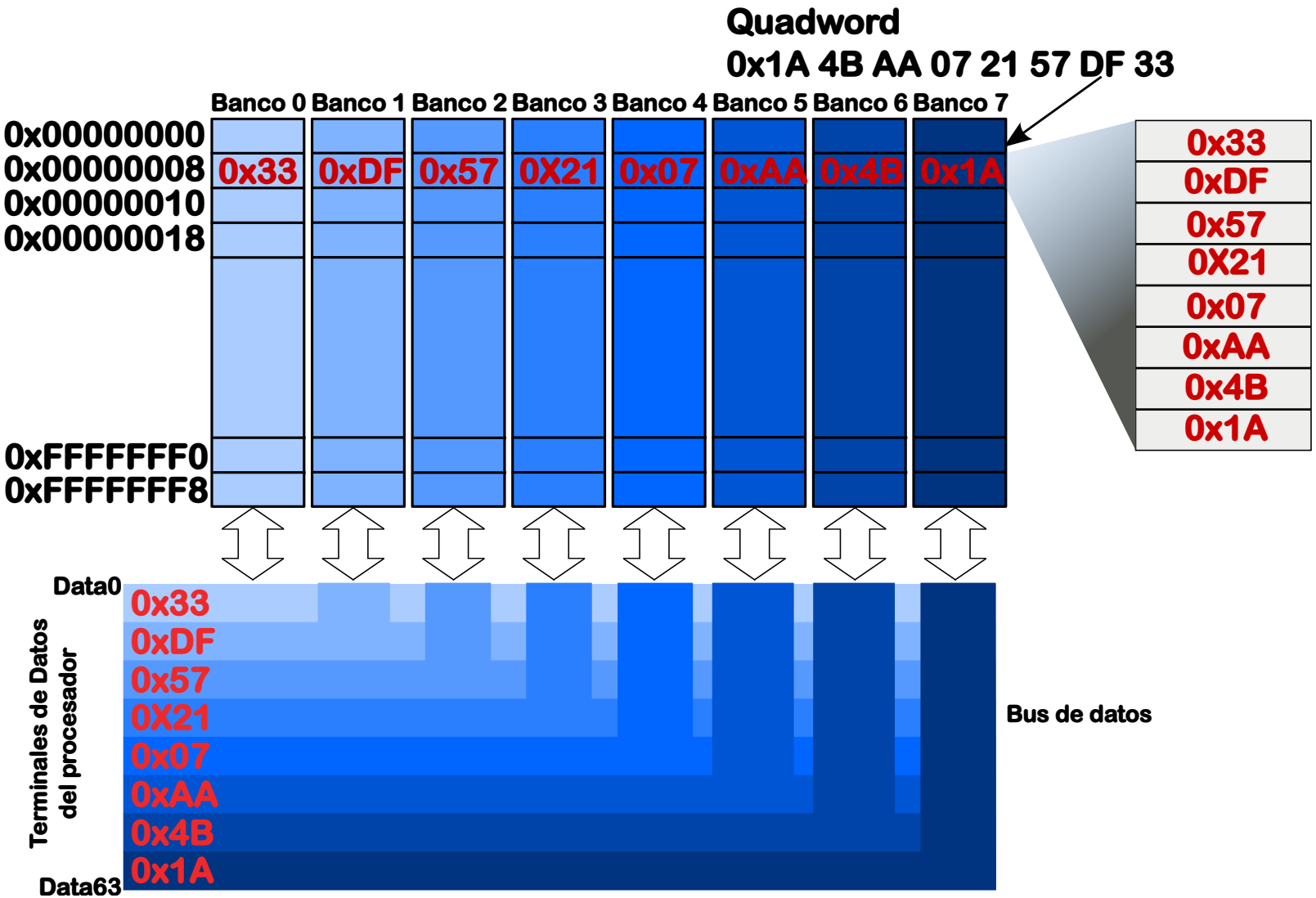
¿Estaba de verdad “al derecho”?



Probemos almacenarlo “al reves”



llegó “al derecho” . . . para la CPU



Endianess vs. Hardware

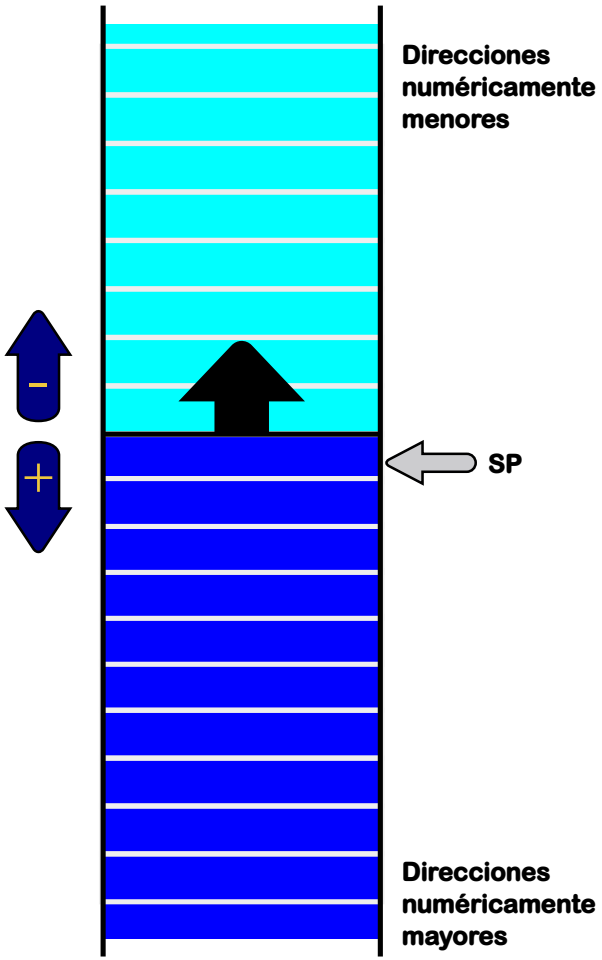
- ¿Quedó claro?. En general es el Ingeniero de Hardware quien define el orden de los bytes que deben llegar al Procesador.
- La verdad es que la tendencia es Little endian a nivel de hardware.
- No obstante ARM permite manejar el endianness de manera configurable y puede variar cuando se trata de Sistema Operativo o de aplicaciones.
- Lo usual es usar Little Endian.
- Por default ARM mapea las direcciones en donde busca código en Little Endian.
- Las variantes que maneja para configurar Big Endian o Litle Endian aplican a espacios de memoria en donde almacena datos.

Registros de propósito general y dedicados

- En el Nivel de Aplicación una CPU ARMv7, presenta 13 registros de proposito general idénticos en funcionalidad.
- Se llaman **`R0`**, **`R1`**, **`R2`**, ...**`R12`**
- Hay tres registros que son en principio registros dedicados, y que se pueden denominar **`R13`**, **`R14`**, y **`R15`**.
- **SP: Stack Pointer**. Es el mismo registro **`R13`**. ARM considera su uso solo como puntero a la pila.
- **LR: Link Register**. Es el mismo registro **`R14`**. Guarda la dirección de retorno cuando se producen branches (se evita guardarla en la pila para acelerar la ejecución de la instrucción de retorno.
- **PC: Program Counter**. Es el puntero a la próxima instrucción a buscar. En Modo Thumb no hay forma de modificarlo. Solo se lo puede hacer ejecutando en modo ARM.

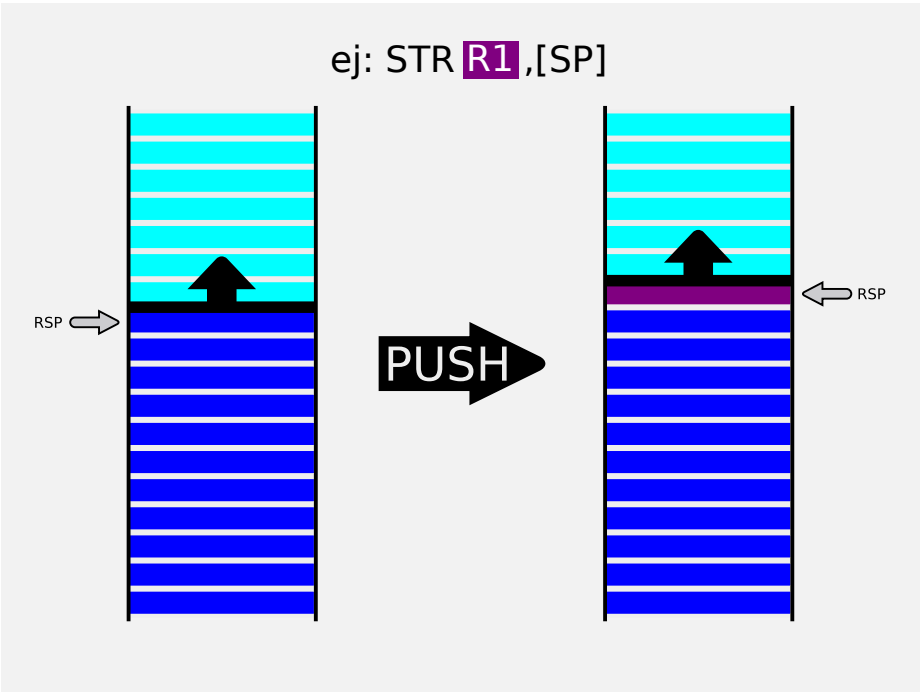
Funcionamiento básico

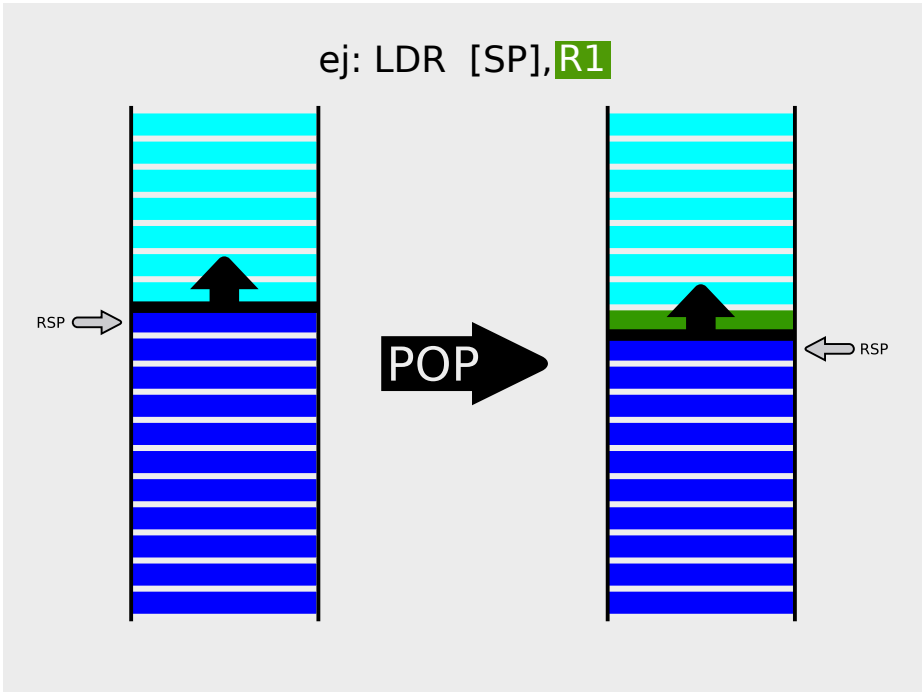
Pila



- La pila (stack) es un área de memoria contigua, que se maneja como una cola **LIFO** (**L**ast **I**n **F**irst **O**ut)
- El tamaño de este bloque puede ser tan grande como se necesite.
- El bloque de memoria se recorre mediante un registro de propósito general, denominado habitualmente en forma genérica stack pointer, y que en los procesadores ARM es el R13 o SP cuyo tamaño es de 32 bits.

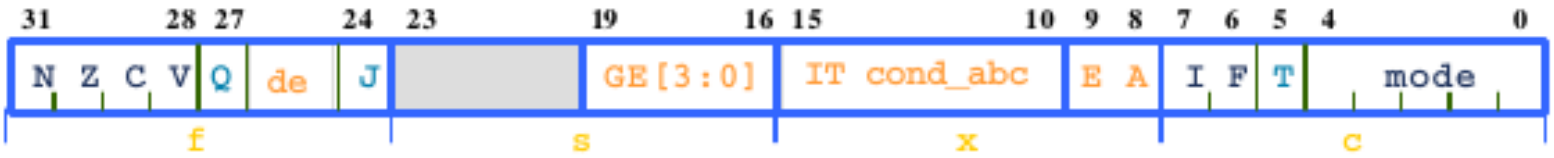
Funcionamiento básico





- La Operación de guardar un dato en una pila se denomina genéricamente **PUSH**, y la de retirarlo **POP**.
- Cuando realiza una operación **PUSH**, el procesador decrementa el registro **`SP`** y luego escribe el dato en la pila, en la dirección apuntada por el registro **`SP`**.
- Cuando realiza una operación **POP**, el procesador lee el ítem apuntado por el registro SP, y luego incrementa éste último registro.

(Current/Saved) Program Status Register



- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE and later only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ and later only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- T Bit
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
  - Introduced in Architecture 4T
- Mode bits
  - Specify the processor mode
- New bits in V6
  - GE[3:0] used by some SIMD instructions
  - E bit controls load/store endianness
  - A bit disables imprecise data aborts
  - IT [abcde] IF THEN conditional execution of Thumb2 instruction groups

Generalidades

- Instrucciones de tamaño fijo (RISC). 32 bits en estado ARM. Se ejecutan en un solo ciclo de clock
- Arquitectura Load/Store (RISC)
  - No se manipula de manera directa el contenido de la memoria
  - Los datos en memoria se llevan a la CPU se procesan utilizando los registros, y luego se regresan los resultados a la dirección apropiada de memoria.
- Los Cores pueden estar en estado ARM o estado Thumb
  - Esto determina que set de instrucciones se está utilizando
  - La transición entre ambos estados es por software (Instrucciones específicas)
- Ejecución condicional (reduce el uso de branching)
  - Diferencias de uso entre estados ARM y Thumb

Acceso a memoria: Load Store

```
1  LDR r0,[r1] /*Carga en R0, una word en memoria direccionada
       por R1*/
2  STR r0,[r1] /*Almacena R0 en memoria direccionada por R1*/
3  LDR r4,[r0+960] /*Carga en r4, una word en memoria
       direccionada por R0+960*/
4  STR r8,[r0],4 /* Almacena R8 en memoria direccionada por R0 y
        le suma 4 a R0*/
```

Ejecución condicional

Este código:

```
1    CMP  r0, #5
2    BEQ  BYPASS
3    ADD  r1, r1, r0
4    SUB  r1, r1, r2
5 BYPASS
6     ...
```

Se puede escribir de este modo:

```
1    CMP     r0, #5
2    ADDNE   r1, r1, r0
3    SUBNE   r1, r1, r2
```

Se implementa en el código de operación. Los 4 bits mas significativos del OPCODE se matchean con los del registro cpsr. Si la AND es FALSE ejecuta sino hace un NOP

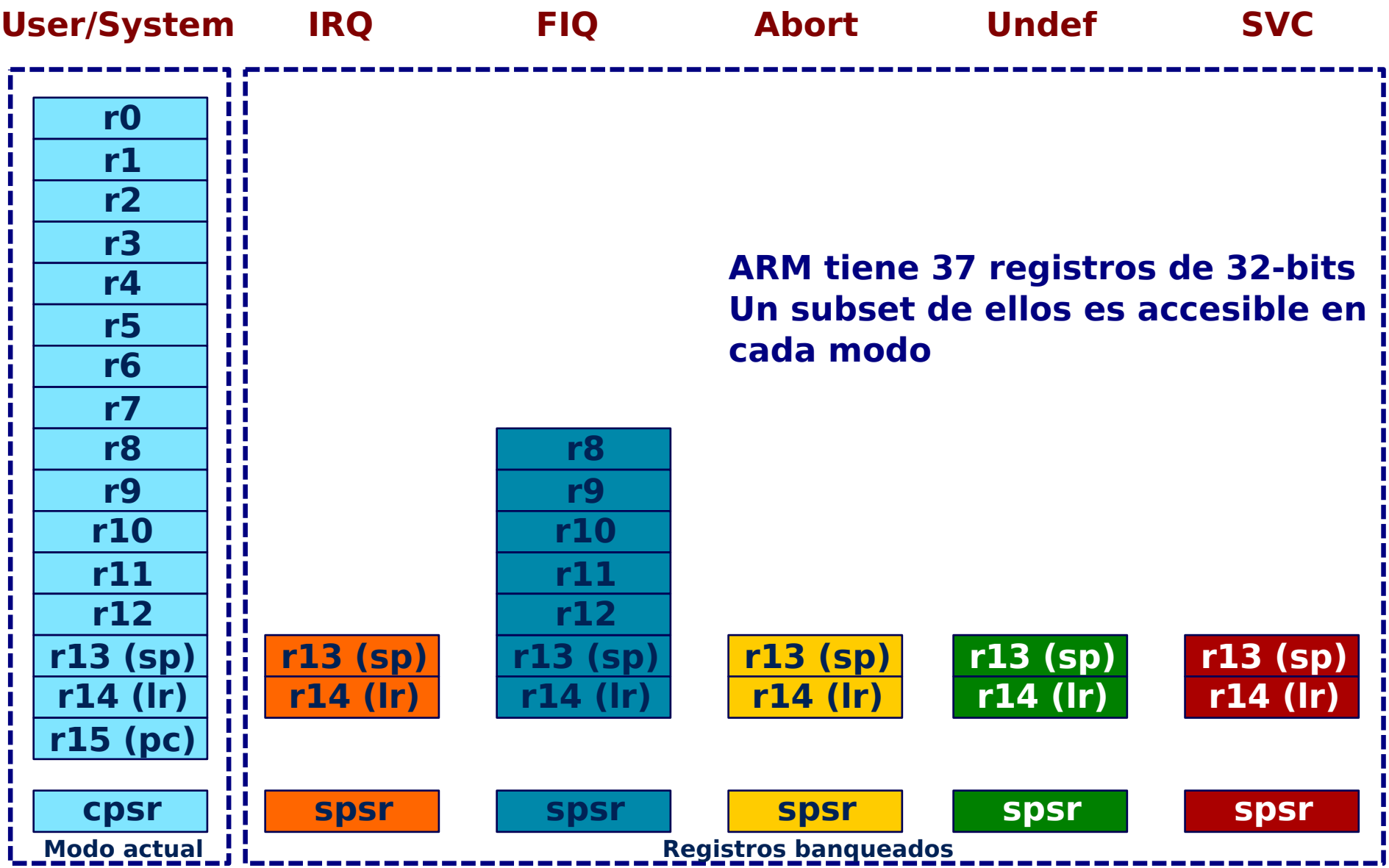
Condiciones

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

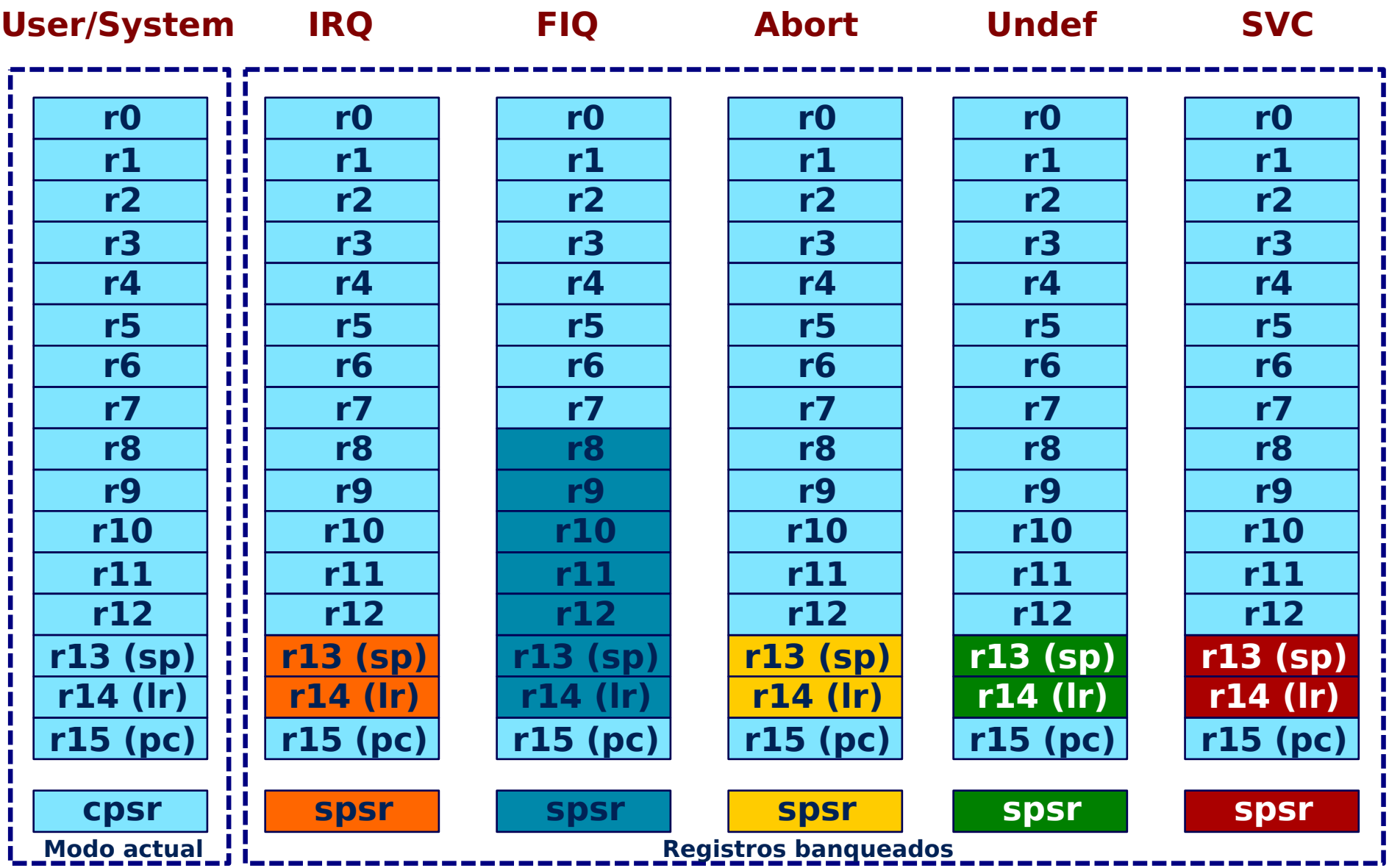
Modos del procesador para CORTEX-A y CORTEX-R

Modo	Descripción	
SVC	**Modo Supervisor**. Se ingresa por interrupción de software (SWI), o luego de un reset.	Modos Privilegiados
FIQ	**Fast Interrupt**. Se ingresa por interrupción de hardware de alta prioridad (fast).	
IRQ	**Interrupt**. Se ingresa por interrupción de hardware de prioridad normal.	
Abort	Se ingresa para manejar violaciones de acceso a memoria.	
Undef	Utilizado para manejar instrucciones indefinidas.	
System	Es el modo Privilegiado y usa los mismos registros que el modo usuario.	
User	Modo en el cual ejecutan la mayoría de las aplicaciones del un S.O.	Modo No Privilegiado

Registros


User/System
IRQ
FIQ
Abort
Undef
SVC
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
Modo actual
ARM tiene 37 registros de 32-bits
Un subset de ellos es accesible en cada modo
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
spsr
spsr
spsr
spsr
spsr
Registros banqueados

Registros


User/System
IRQ
FIQ
Abort
Undef
SVC
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
spsr
Modo actual
Registros banqueados