uc3m

Departamento Tecnología Electrónica

Tema 3: Arquitectura Interna de una CPU

Sistemas Digitales Basados en Microprocesadores

Universidad Carlos III de Madrid

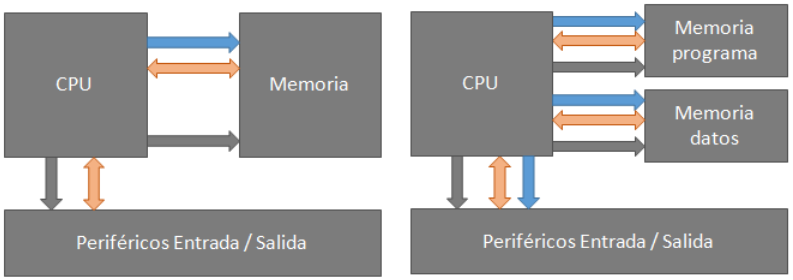
Dpto. Tecnología Electrónica

Índice

1 - Tipos de Arquitecturas y familia ARM

Tipos de Arquitecturas

- Desde el punto de vista de los Registros Internos:
  - Basadas en Acumulador
  - Basadas en Registros
- Desde el punto de vista de los Buses Internos:
  - Von Neumann
  - Harvard
- A partir de una arquitectura Harvard, se puede generar una Arquitectura Von Neumann
  - Fusionando buses
  - Usando una misma memoria principal para datos y programa

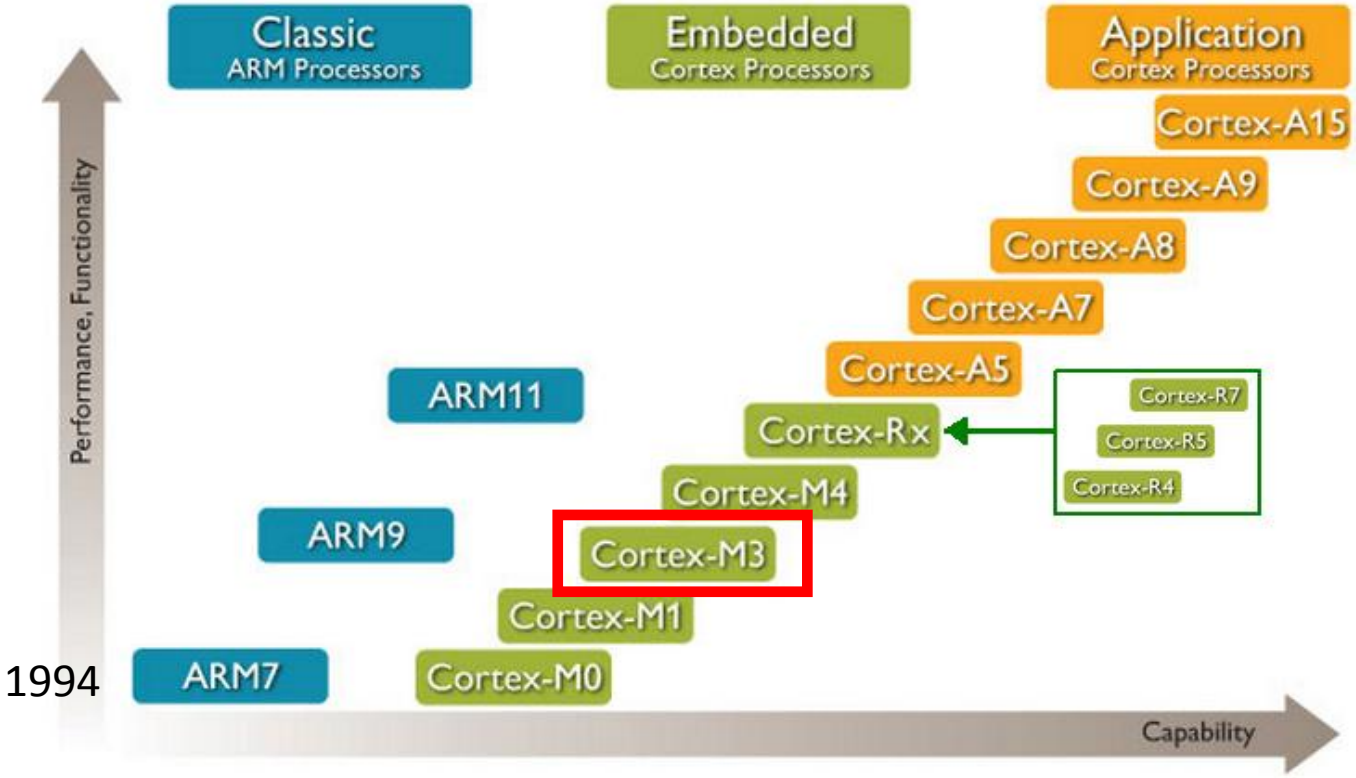


La Familia ARM: Origen

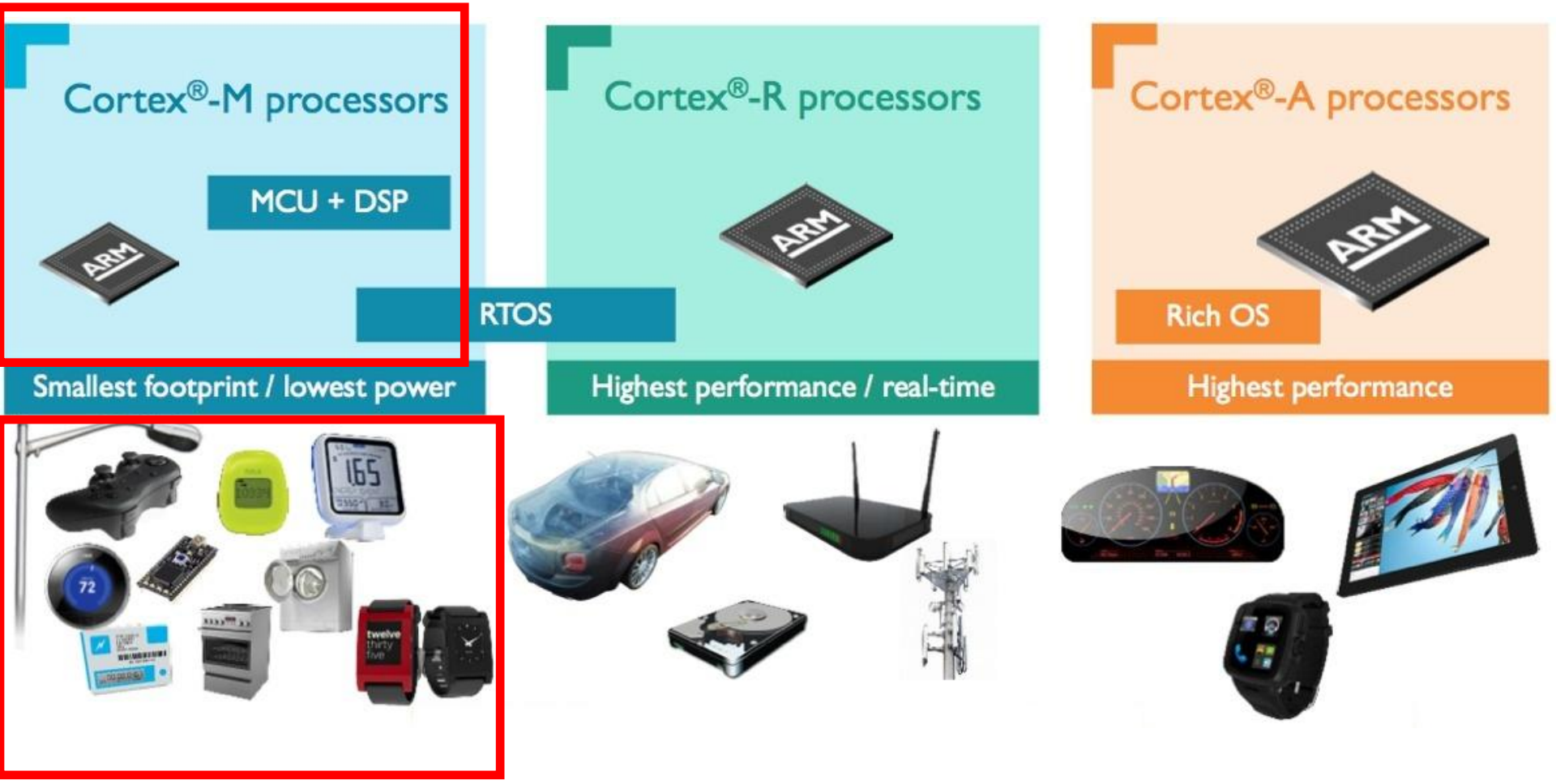
- ARM Holdings plc es una multinacional dedicada a los semiconductores y al desarrollo de software con sede en Cambridge, Reino Unido.
- Es considerada la empresa dominante en el campo de los chips de smartphones y tablets.
- La compañía fue fundada como Advanced RISC Machines (ARM) en 1990 por Robin Saxby
  - Una empresa conjunta entre Acorn Computers (desde 1983), Apple Computer (ahora Apple Inc.) y VLSI Technology.
- Pretende promover el desarrollo de la arquitectura ARM, la cual fue utilizada en su origen en el Acorn Archimedes y fue seleccionado por Apple para su proyecto Apple Newton.
- Cronología:
  - 1981 – ARM1
  - 1986 – ARM2: primera versión comercial (32 bits), con mejor rendimiento que el 286
  - 1991 – ARM6: Apple Newton y RiscPC
  - 1994 – ARM7TDMI: millones de unidades en teléfonos móviles y videojuegos (2009)

La Familia ARM al completo

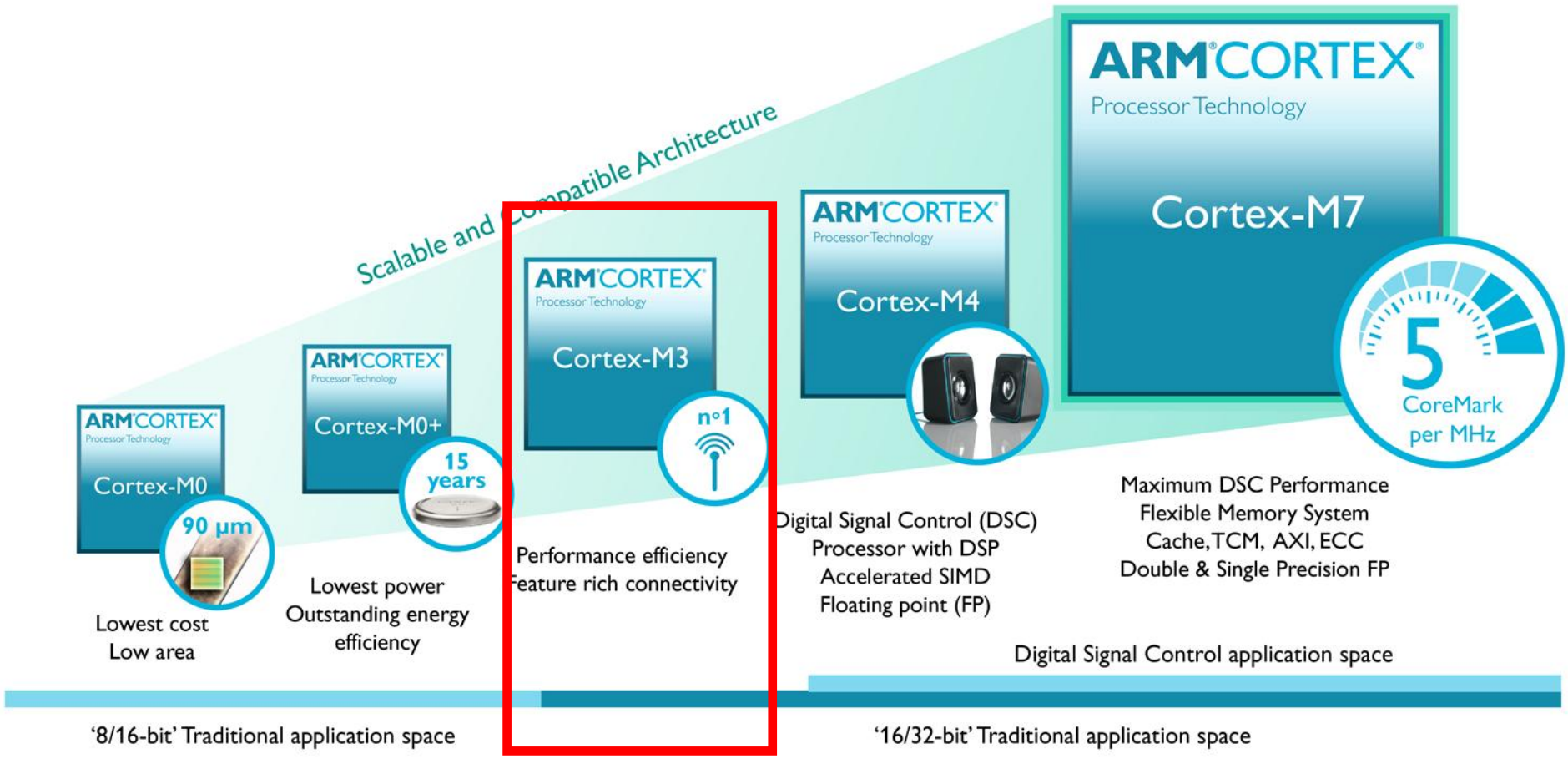
- Todas las soluciones ARM están diseñadas de tal forma que se maximice su rendimiento, reduciendo su consumo:
  - https://www.youtube.com/watch?v=7LqPJGnBPMM
- Tienen distintas versiones para distintos objetivos



La Familia ARM M, ARM R y ARM A



La Familia ARM M



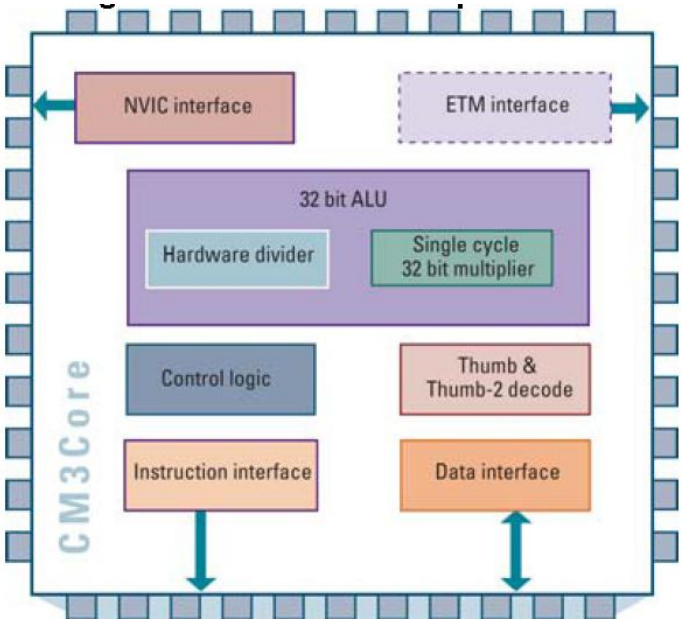
La Familia ARM: Comparativa en funciones

ARM Cortex-M instruction variations

ARM Core	Cortex M0[1]	Cortex M0+[2]	Cortex M1[3]	Cortex M3[4]	Cortex M4[5]	Cortex M7[6]	Cortex M23[7]	Cortex M33
Thumb-1 instructions	Most	Most	Most	Entire	Entire	Entire	Most	Entire
Thumb-2 instructions	Some	Some	Some	Entire	Entire	Entire	Some	Entire
Multiply instructions	32-bit result	32-bit result	32-bit result	32-bit result 64-bit result	32-bit result 64-bit result	32-bit result 64-bit result	32-bit result	32-bit result 64-bit result
Divide instructions	No	No	No	Yes	Yes	Yes	Yes	Yes
Saturated instructions	No	No	No	Some	Yes	Yes	No	Yes
DSP instructions	No	No	No	No	Yes	Yes	No	Optional
Floating-point instructions	No	No	No	No	Optional: SP	Optional: SP or SP & DP	No	Optional: SP
TrustZone instructions	No	No	No	No	No	No	Optional	Optional
Instruction pipeline	3 stages	2 stages	3 stages	3 stages	3 stages	6 stages	2 stages	3 stages
Computer architecture	Von Neuman	Von Neumann	Von Neumann	Harvard	Harvard	Harvard	Von Neumann	Harvard
ARM architecture	ARMv6-M[9]	ARMv6-M[9]	ARMv6-M[9]	ARMv7-M[10]	ARMv7E-M[10]	ARMv7E-M[10]	ARMv8-M[14]	ARMv8-M[14]

Arquitectura del Cortex-M3

- El Cortex-M3 es una Arquitectura Harvard
- Pero a nivel de uso, el Cortex-M3 fusiona los dos mapas de memoria, convirtiendo el CM3Core en equivalente a una Arquitectura Von Neumann:
  - Bus direcciones de 32 bits
    - ***Se direccionan en bytes***
  - Bus de datos de 32 bits
  - 23 registros
    - 13 de propósito general
    - 2 de puntero de pila (SP)
    - 1 PC
    - 1 registro de enlace (LR)
    - 1 registro de estado
    - 4 registros especiales



Arquitectura del Cortex-M3: arranque de la CPU

- La CPU tiene que ser un elemento determinista, por lo que hay que garantizar que sus condiciones iniciales son siempre las mismas:
  - Tiene que tener un valor estable en el SR
  - Tiene que saber el valor inicial del PC
  - Tiene que saber el valor inicial del SP, si existe
  - Y en muchos casos el valor inicial de otros muchos registros
- Dependiendo de la CPU esos valores son fijos, o pueden ser modificados por el programador
  - En ese caso se graba el valor en una posición de memoria que al inicio es accedida por la CPU para obtener su valor (vector de reset, etc.)

2 - Notación RTL

Notación RTL

- El *Register Transfer Language* (RTL) es un lenguaje que simboliza la transferencia de datos entre registros
- Se utiliza, entre otras cosas, para representar el funcionamiento de un determinado proceso, por ejemplo:
  - La ejecución de una instrucción
  - El funcionamiento de la Unidad de Control
  - La secuencia de pasos que conlleva una interrupción
  - Etc.
- Las variantes son muchas, por lo que se pueden encontrar especificaciones en RTL muy distintas de unos autores a otros
  - Mientras haya consistencia entre las representaciones, se puede considerar como válida cualquiera.

Notación RTL

- En este curso se va a utilizar una representación muy simplificada:
  - Paso del contenido de un registro a otro:
    - MAR $\Leftarrow$ PC ; lleva el contenido del PC al MAR
  - Paso del contenido de la memoria a un registro -> Operación de lectura
    - MBR $\Leftarrow$ (MAR) ; lleva el contenido que tiene la memoria en la posición indicada por MAR, al registro MBR
  - Paso del contenido de un registro a memoria -> Operación de escritura
    - (MAR) $\Leftarrow$ MBR ; lleva el contenido del registro MBR a la posición indicada en memoria por MAR
  - Operaciones con el operando fuente sin más
    - PC $\Leftarrow$ PC + 1 ; incrementa el PC en una unidad

3 - La Unidad de Control

La Unidad de Control

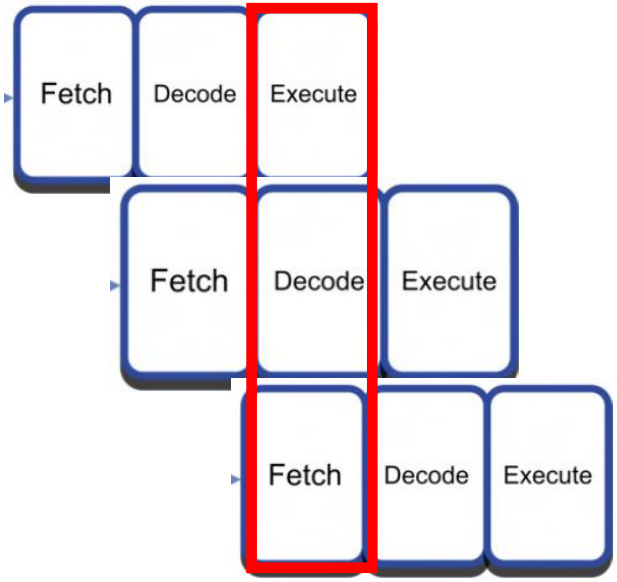
- Es el elemento que se encarga de gestionar el funcionamiento de toda la CPU
- Se trata de un autómata que se basa en ejecutar continuamente tres fases:
  - ***Fetch*** (captura): Se captura la instrucción de la memoria. Esto conlleva las siguientes operaciones:
    - MAR ⇐ PC ; ; Contador de programa en el registro MAR
    - PC ⇐ PC + 1 ; Aumentar el contador del programa
    - MBR ⇐ (MAR) ; Lectura de la posición de memoria con la instrucción
    - IR ⇐ MBR ; Transferencia de la instrucción al registro IR
  - ***Decode*** (decodificación): Se interpreta el contenido del IR y se prepara a la CPU para la siguiente fase
  - ***Execute*** (ejecución): Se ejecutan los pasos necesarios para la consecución de la operación

La Unidad de Control

- Al tratarse de un autómata, funcionará a un ritmo marcado por el reloj del sistema:
  - Es importante tener en cuenta que la ejecución de una instrucción conlleva un consumo de tiempo, no es cero.
  - En realidad se habla de tiempo marcado por ***ciclos máquina***
    - Dependiendo de la CPU, cada ciclo máquina puede ser un ciclo de reloj, o puede ser un número fijo de ciclos de reloj
  - Por tanto, cada instrucción consumirá un determinado número de ciclos máquina en la ejecución de sus tres fases.
    - La fase de fetch es común a todas las instrucciones
    - El número de ciclos en las otras dos fases puede ser distinto para cada instrucción, dependiendo de lo que haya que hacer
  - Cuanto mayor frecuencia pueda tener el reloj del sistema, más rápida será la ejecución del programa
- La Unidad de Control genera tantas señales internas como necesite para controlar cada uno de los componentes de la CPU (**bus de control interno**)

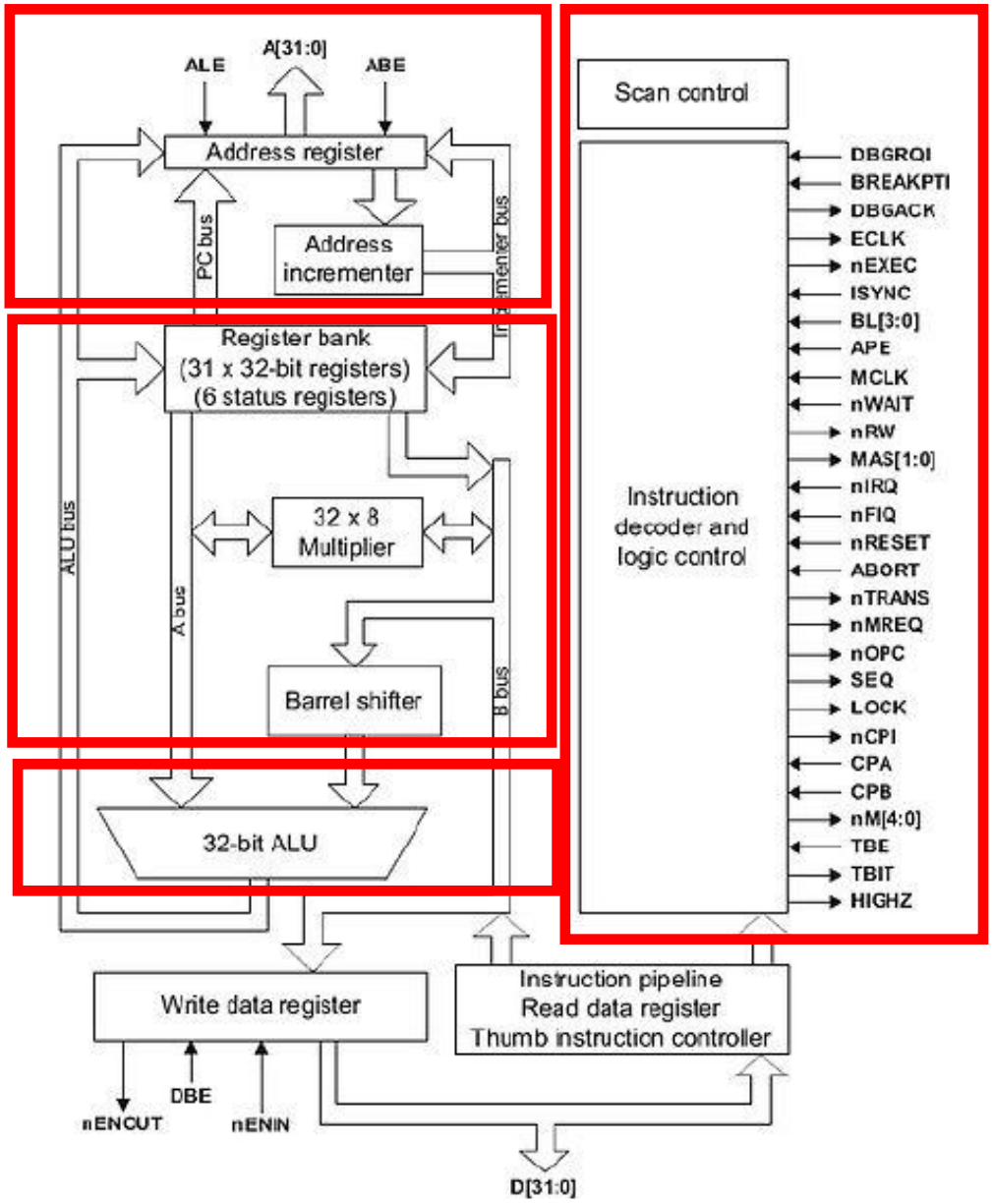
La Unidad de Control

- Dependiendo de la Arquitectura de la CPU, el diseñador de la misma puede haber hecho una optimización de recursos internos, para procesar varias instrucciones en “paralelo”:
  - Por ejemplo, mientras se está ejecutando una instrucción, se puede estar capturando la siguiente (pre-fetch)
  - A esta técnica se le denomina ***segmentación o pipeline*** y el número de etapas (segmentos) puede variar de una a otra
  - El Cortex-M3 presenta una segmentación a 3 niveles:
    - Fetch
    - Decodificación
    - Ejecución

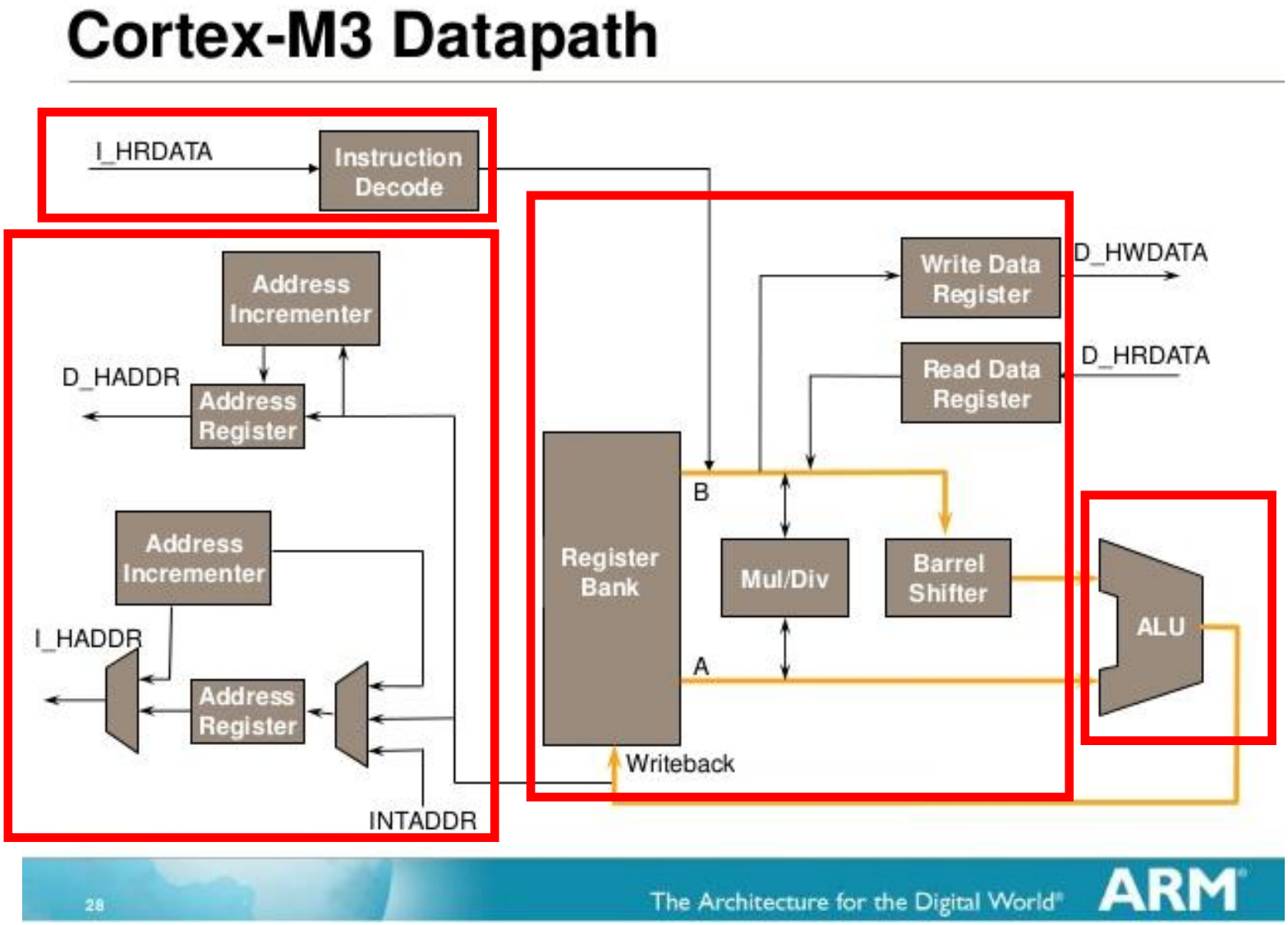


4 - La Unidad Aritmético Lógica y la Ruta de Datos

La Ruta de Datos del ARM7

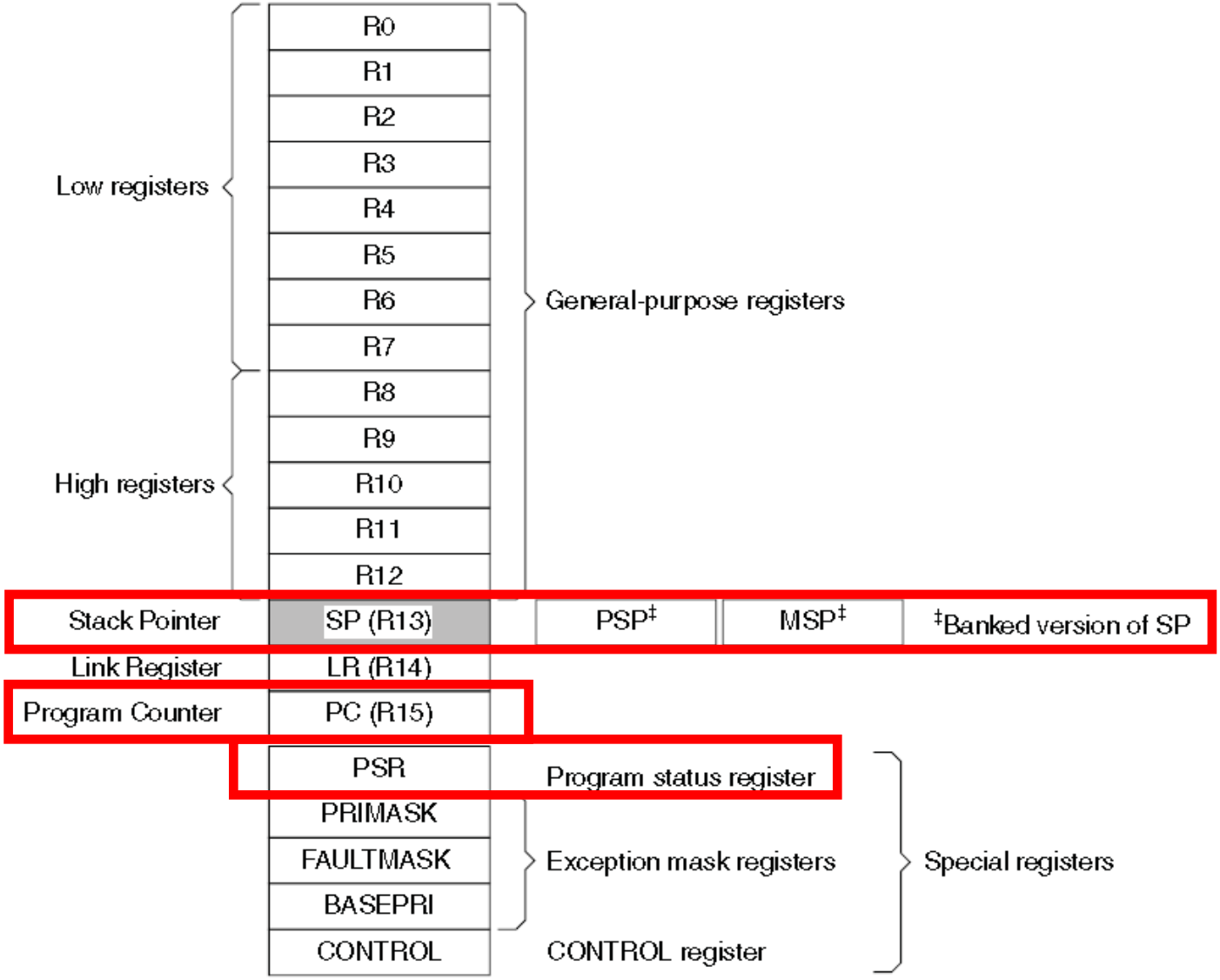


La Ruta de Datos del Cortex-M3



5 - Los Registros de la CPU

Registros Internos del Cortex-M3



El Contador de Programa (PC)

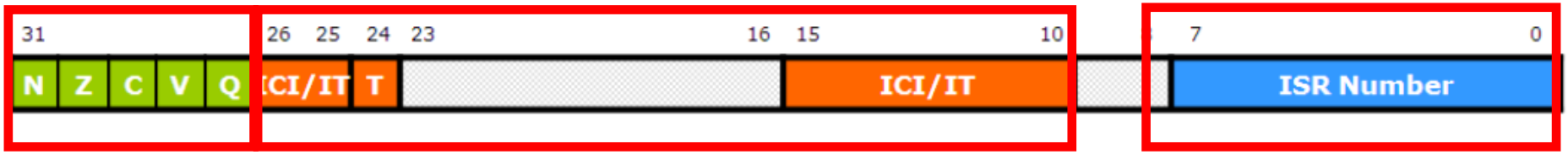
- El contador del programa contiene en cada instante la dirección de la **siguiente instrucción** a capturar
- En el Cortex-M3, el contador de programa es un registro con las siguientes características:
  - Mismo tamaño que el bus de direcciones
  - Capacidad de ser modificado completamente por determinadas instrucciones
  - Capacidades especiales de "autoincremento" para no saturar la ALU en cada fase de *fetch* y captura de parámetros de la instrucción (caso de más de una palabra por instrucción)

El Registro de Estado (SR)

- Sirve para almacenar el estado en el que se encuentra la CPU tras la ejecución de una instrucción
- El estado se indica con una serie de flags, que son cada uno de los bits de ese registro de estado
- Los flags típicos de un registro de estado son:
  - Z: se activa si la última operación ha dado como resultado un 0
  - N: se activa si la última operación ha dado como resultado un número negativo
  - C: se activa si la última operación ha producido acarreo
  - V: se activa si la última operación ha dado desbordamiento
- También puede haber otros flags que indiquen interrupción, etc. (o que habiliten esos eventos)

El Registro de Estado del Cortex-M3

- En el Cortex-M3 se tienen 3 registros de estado, que se combinan en el PSR:
  - Application Program Status Register (APSR)
    - Que contiene los flags de estado
  - Interrupt Program Status Register (IPSR)
    - Que contiene información respecto a la RAI en ejecución
  - Execution Program Status Register (EPSR)
    - Que contiene información de ejecución del programa



El Puntero de Pila (SP)

- En las CPUs actuales, es común disponer de un registro adicional, denominado Puntero de Pila (SP)
  - Es un registro que contiene una dirección de memoria RAM
  - Esa dirección se incrementa o decrementa según se hagan operaciones con “la pila”
- La Pila:
  - Es una zona de memoria dedicada a almacenar información en formato LIFO (el último que entra es el primero que sale)
  - Se implementa en memoria RAM, y puede crecer hacia direcciones altas o hacia direcciones bajas
  - El SP puede apuntar a la última dirección escrita, o a la primera libre
    - Por ejemplo, si crece hacia direcciones más bajas, y apunta a la primera dirección libre, una operación de escritura en la pila (PUSH) sería:

      $(SP) \Leftarrow MBR$; $SP \Leftarrow SP-1$
    - Y una operación de lectura en la pila (PULL) sería

      $SP \Leftarrow SP+1$; $MBR \Leftarrow (SP)$

El Puntero de Pila del Cortex-M3

- El Cortex-M3 implementa 2 pilas, la *main stack* y la *process stack*:
  - Cada una tiene su propio SP

- El Cortex-M3 utiliza pilas completas descendentes y el SP apunta al último elemento introducido en la pila, es decir, cuando se introduce un nuevo elemento en la pila, se decrementa el SP y luego se introduce el valor

6 - La Memoria Principal

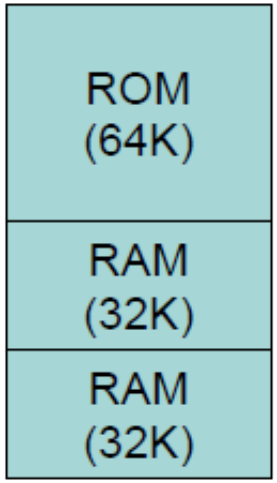
La Memoria Principal

- Manteniendo la filosofía de Von Neumann, la memoria almacena tanto datos como instrucciones
  - Las instrucciones y datos permanentes deberían estar en memoria no-volátil (ROM, EPROM, EEPROM, etc.)
  - Los datos intermedios deberían estar en memoria volatil (RAM)
- Normalmente se utilizan distintos chips de memoria y se guardan instrucciones y datos en distintas partes del **mapa de memoria:**
  - Distintos chips que comparten líneas de datos y de direcciones
    - Cada chip de una tecnología, según necesidades
  - Hace falta una selección del chip al que se accede en el mapa de memoria, dependiendo del rango al que pertenece la dirección -> **Decodificador de direcciones**
- Los datos a manejar, las direcciones a acceder y las instrucciones a ejecutar en la memoria se disponen en los diferentes **buses** que acceden ala misma: bus de datos, bus de direcciones y bus de control

La Memoria Principal – Mapa de memoria

• Indica las direcciones de comienzo y fin de cada tipo o chip de memoria de la memoria completa. Se puede expresar en:

-> Hexadecimal: Para el mapa de memoria en sí
-> Binario: Para diseñar el codificador de direcciones



Address (hex)	Address (bin) A16	A15	A14…A0
00000h	0	0	0…0
0FFFFh	0	1	1…1
10000h	1	0	0…0
17FFFh	1	0	1 1
18000h	1	1	0…0
1FFFFh	1	1	1…1

La Memoria Principal – Decodificador de direcciones

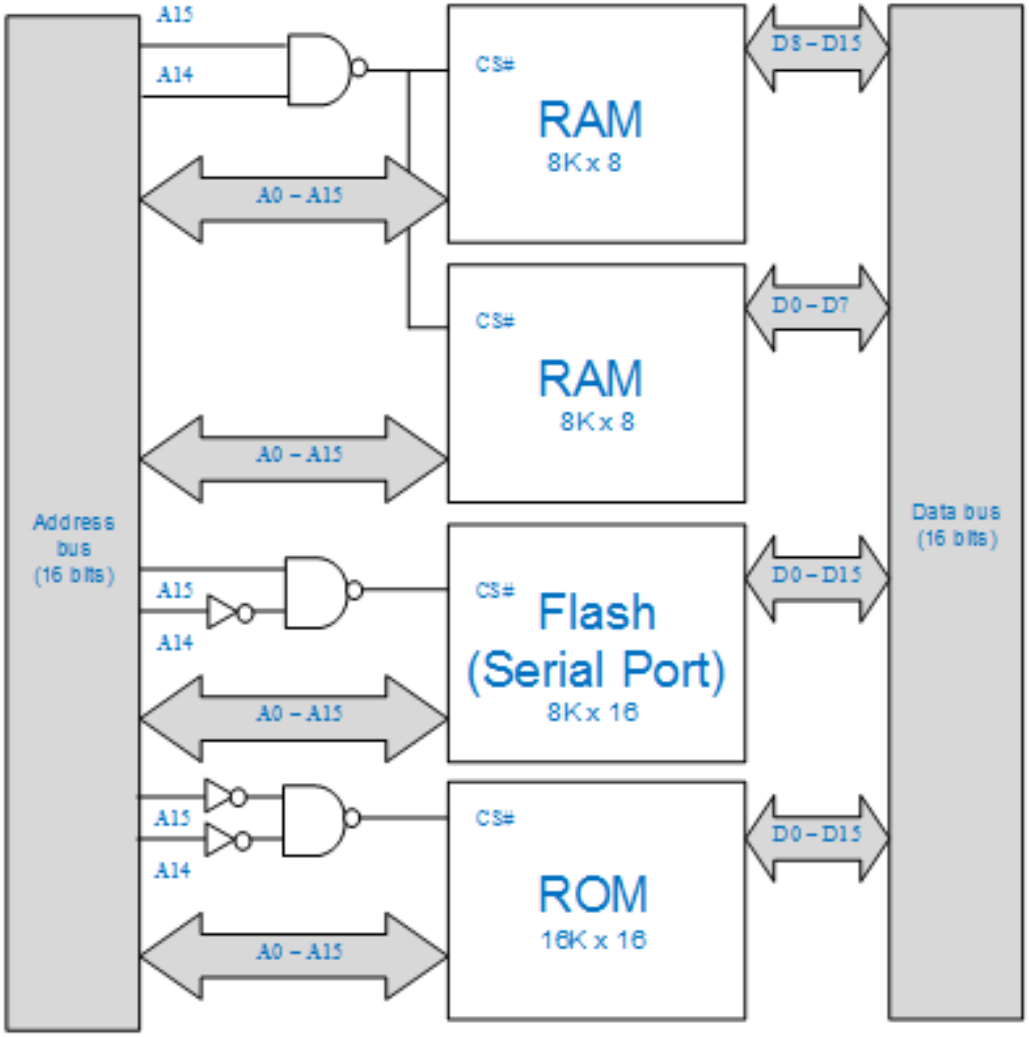
- Su misión es activar señales de selección de dispositivos (CS#) dependiendo del rango en el que se encuentre la dirección de memoria accedida
- Para construirlo se pueden utilizar distintas estrategias:
  - Completa: cada CS# se obtiene utilizando todas las líneas de direccionamiento del micro. Hay una relación biunívoca entre dirección física accedida y contenido del bus de direcciones
  - Parcial: no todas las líneas de direcciones forman parte del proceso de decodificación. Existen direcciones físicas a las que se puede acceder con distintos contenidos del bus de direcciones
  - Por Bloque: se van definiendo bloques de direcciones y luego esos bloques se vuelven a decodificar (es una decodificación por pasos, o jerárquica). Se suele usar cuando hay bancos de memoria adicionales.

La Memoria Principal - Buses

- La comunicación de la información necesaria para trabajar con la memoria se hace a través de buses (externos)
  - **Bus de Datos:** el que lleva el dato al que se está accediendo
    - Tiene el mismo número de líneas que el tamaño de la palabra de la CPU (aunque puede haber excepciones)
  - **Bus de Direcciones:** el que lleva la dirección del dato al que se quiere acceder
    - Tiene el mismo tamaño que el MAR y el PC
    - Cuanto más líneas se tenga, mayor será la capacidad de memoria que se le pueda conectar a la CPU
  - **Bus de Control:** el que contiene las distintas líneas que hacen posible un protocolo de comunicación con los chips de memoria
    - Por ejemplo, la señal de R/W, de AS#, o de CS#

La Memoria Principal

Ejemplo completo con mapa de memoria, decodificador de direcciones y buses

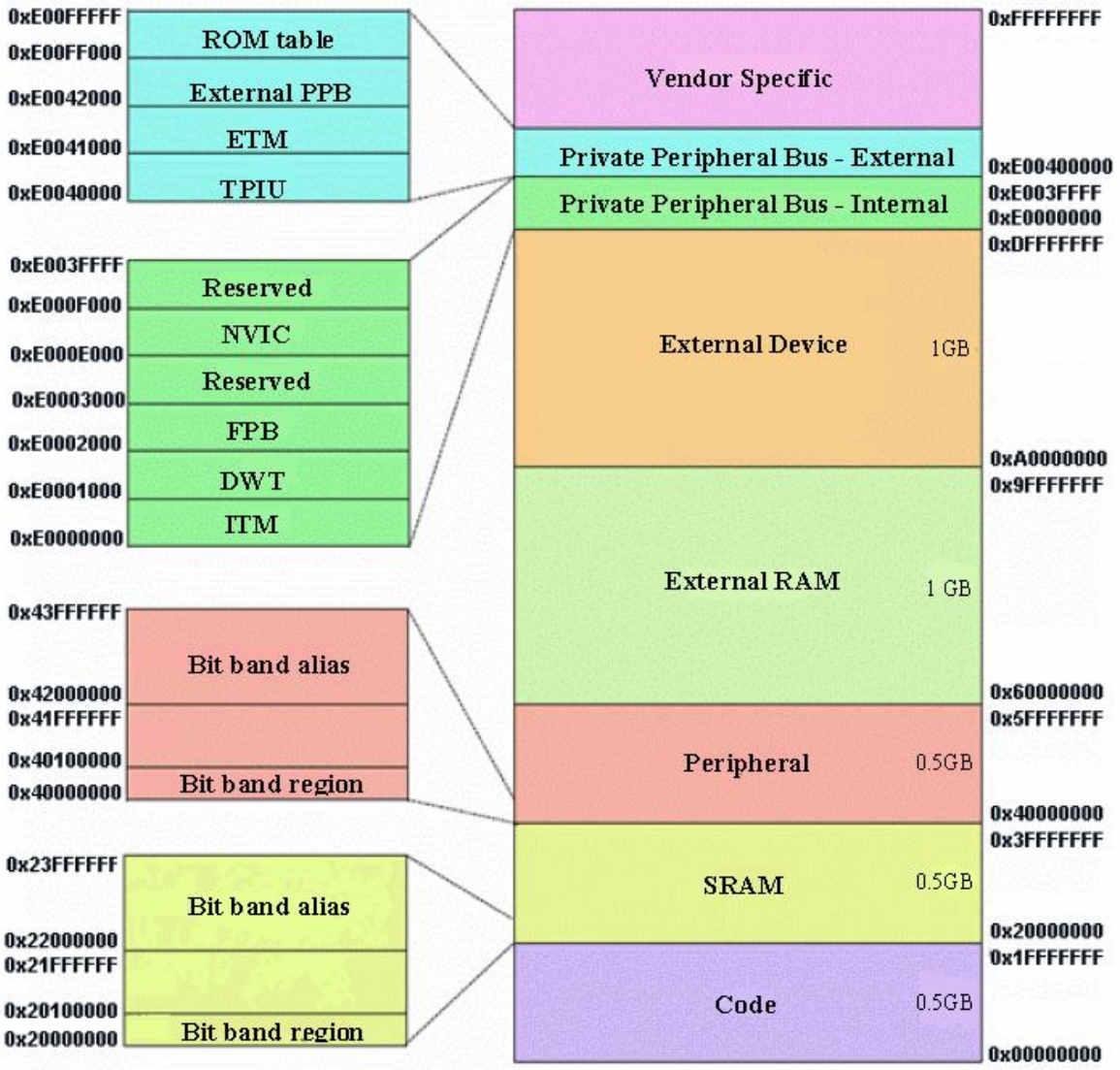


La Memoria Principal – Bancos de memoria

- Casi siempre se puede ampliar memoria cuando te quedas sin ella internamente, creando artificialmente bancos de memoria externos
  - Una determinada posición de memoria (un registro) almacena el número de banco con el que se está trabajando
  - Al acceder a la memoria, el decodificador de direcciones toma como una de sus entradas también el valor de esa posición, y decide que chip selecciona
  - Para cambiar de banco sólo hay que volver a escribir en memoria el número del nuevo banco de memoria al que acceder
- De esta forma se obtiene una capacidad de 2^AxB
  - A es el tamaño del bus de direcciones
  - B el número de bancos que se contemplan

La Memoria Principal del Cortex-M3

- El Cortex-M3 describe un modelo de memoria como el de la figura
- Deja espacio para que cada vendedor mapee aquellos recursos que quiera ofrecer
  - Memoria
  - Periféricos
- Se direcciona en bytes (no palabras de 32 bits)



7 – Las Instrucciones

El Registro de Instrucción (IR)

- Es el encargado de mantener la instrucción que se va a ejecutar, para que se decodifique y se emprendan las acciones necesarias
- El tamaño del IR determina el número máximo de instrucciones, por las combinaciones posibles (8 bits = 256 operaciones máximas)
- Sin embargo, las instrucciones se codifican de forma muy estructurada, lo que limita las posibilidades y simplifica la decodificación de las mismas, y el trabajo del programador
  - Como mínimo, las instrucciones se codifican en dos partes:
    - ***Opcode*** o código de operación: es el código que indica la instrucción a ejecutar y su variante (modo de direccionamiento, etc.)
    - ***Parámetros***: normalmente determinan un (o varios) operando o la dirección de un (o varios) operando.

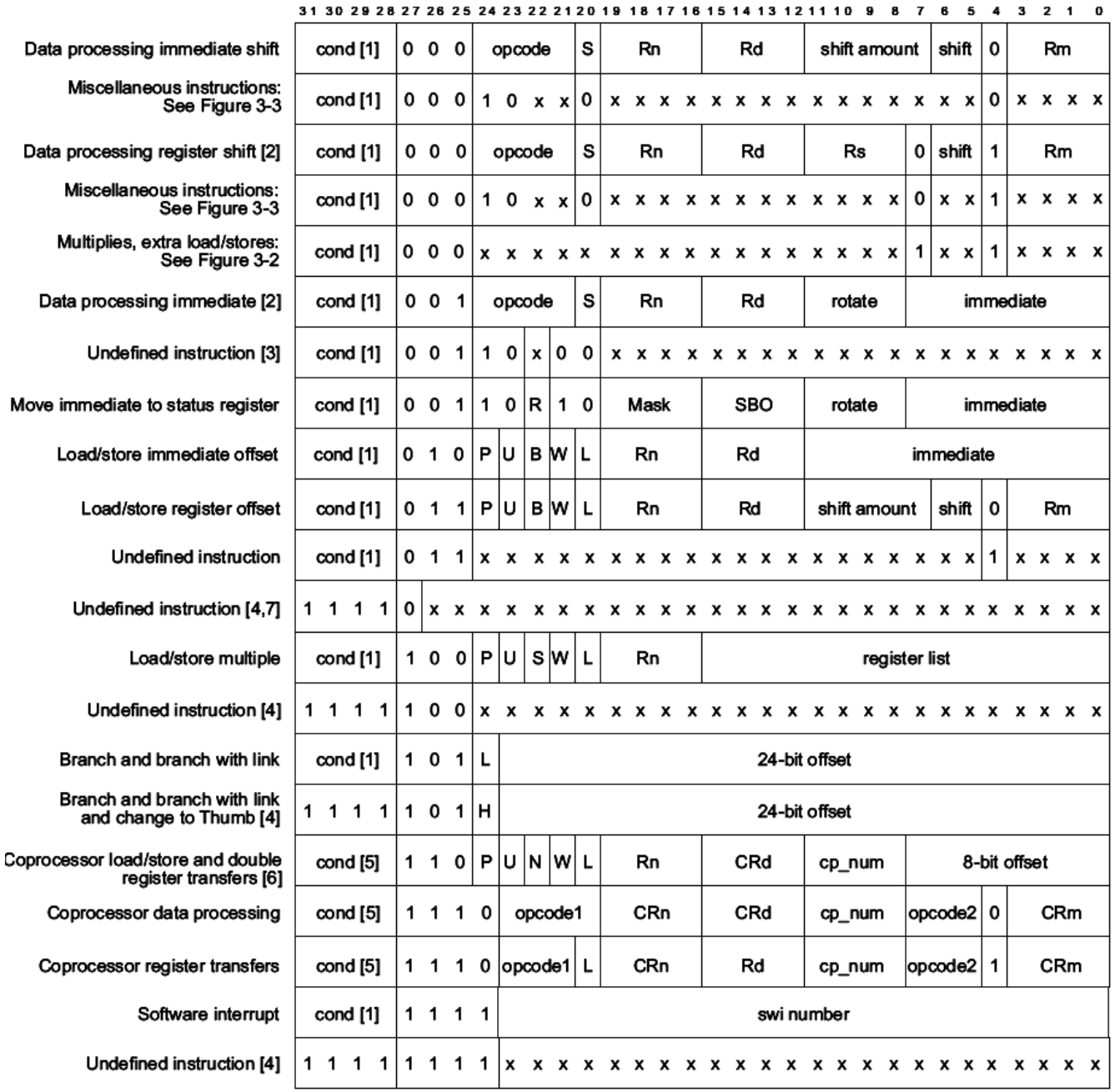
El Registro de Instrucción (IR)

- Respecto al número de instrucciones que contempla una CPU, existen las siguientes arquitecturas:
  - **CISC:** Son CPUs que contemplan un gran número de instrucciones
    - Normalmente son instrucciones complejas, con gran número de variantes
    - Muchas de ellas se utilizan un número muy limitado de veces
    - Facilitan mucho la programación, al contemplar operaciones complejas
  - **RISC:** Un número de instrucciones reducido
    - Instrucciones muy sencillas, normalmente de una única palabra de memoria
    - La CPU es mucho más sencilla (pequeña y barata)
    - La programación se complica, al tener que hacer operaciones no excesivamente complejas, con un gran número de instrucciones sencillas
    - El ARM7TDMI presenta una arquitectura RISC

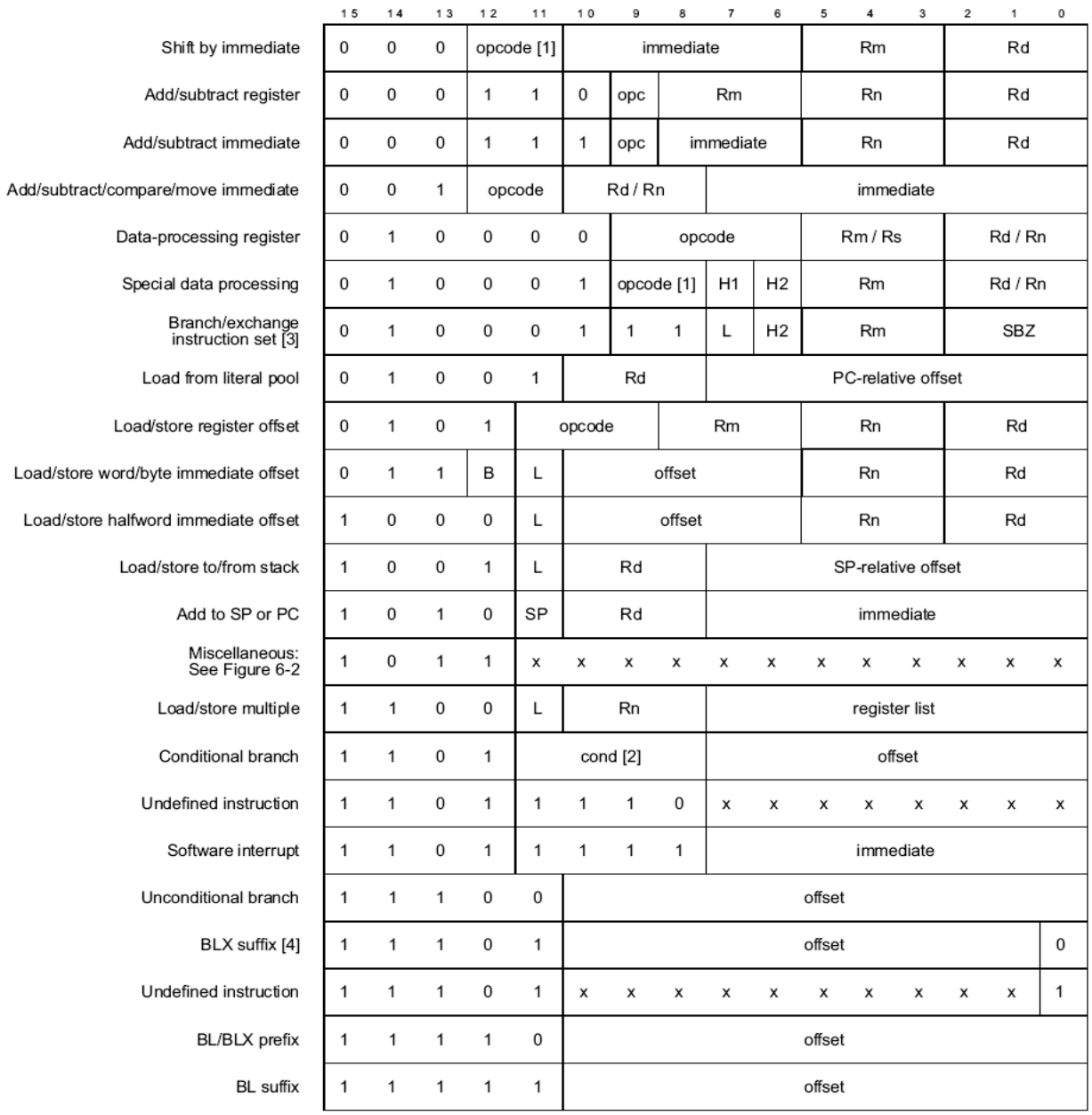
El Registro de Instrucción (IR)

- El IR suele tener el tamaño de una palabra de memoria.
  - Pero las instrucciones pueden ocupar una o varias palabras, no es el caso del Cortex-M3
- Las instrucciones del Cortex-M3 tiene las siguientes características:
  - Existen 2 juegos de instrucciones: Thumb y Thumb-2
    - Aquí se verá solo el Thumb-2 (o Thum16)
  - Todas las instrucciones son de 32 bits (1 palabra)
    - Aunque algunas de las instrucciones se pueden codificar en 16 bits
  - Arquitectura *load-store*
    - Todas las operaciones trabajan sólo con registros, salvo las de transferencia de datos
  - Contemplan 3 operandos (dos fuentes y uno destino)
  - Ejecución condicional de todas las instrucciones

ARM 7: Instrucciones generales



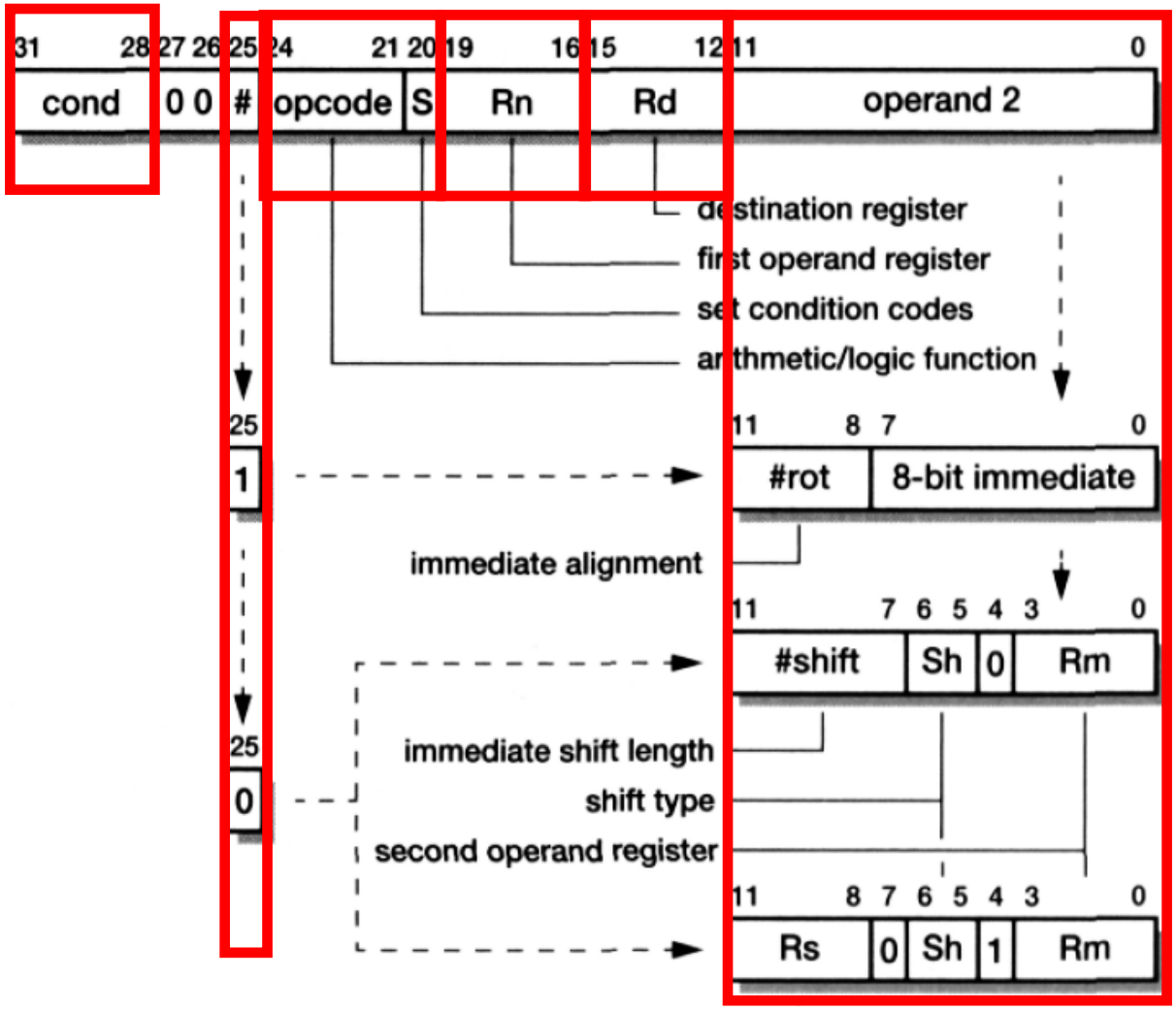
Thumb16: Instrucciones generales



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift by immediate	0	0	0	opcode [1]		immediate					Rm			Rd		
Add/subtract register	0	0	0	1	1	0	opc	Rm			Rn			Rd		
Add/subtract immediate	0	0	0	1	1	1	opc	immediate			Rn			Rd		
Add/subtract/compare/move immediate	0	0	1	opcode		Rd / Rn		immediate								
Data-processing register	0	1	0	0	0	0	opcode				Rm / Rs			Rd / Rn		
Special data processing	0	1	0	0	0	1	opcode [1]		H1	H2	Rm			Rd / Rn		
Branch/exchange instruction set [3]	0	1	0	0	0	1	1	1	L	H2	Rm			SBZ		
Load from literal pool	0	1	0	0	1	Rd			PC-relative offset							
Load/store register offset	0	1	0	1	opcode			Rm			Rn			Rd		
Load/store word/byte immediate offset	0	1	1	B	L	offset					Rn			Rd		
Load/store halfword immediate offset	1	0	0	0	L	offset					Rn			Rd		
Load/store to/from stack	1	0	0	1	L	Rd			SP-relative offset							
Add to SP or PC	1	0	1	0	SP	Rd			immediate							
Miscellaneous: See Figure 6-2	1	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x
Load/store multiple	1	1	0	0	L	Rn			register list							
Conditional branch	1	1	0	1	cond [2]				offset							
Undefined instruction	1	1	0	1	1	1	1	0	x	x	x	x	x	x	x	x
Software interrupt	1	1	0	1	1	1	1	1	immediate							
Unconditional branch	1	1	1	0	0	offset										
BLX suffix [4]	1	1	1	0	1	offset										0
Undefined instruction	1	1	1	0	1	x	x	x	x	x	x	x	x	x	x	1
BL/BLX prefix	1	1	1	1	0	offset										
BL suffix	1	1	1	1	1	offset										

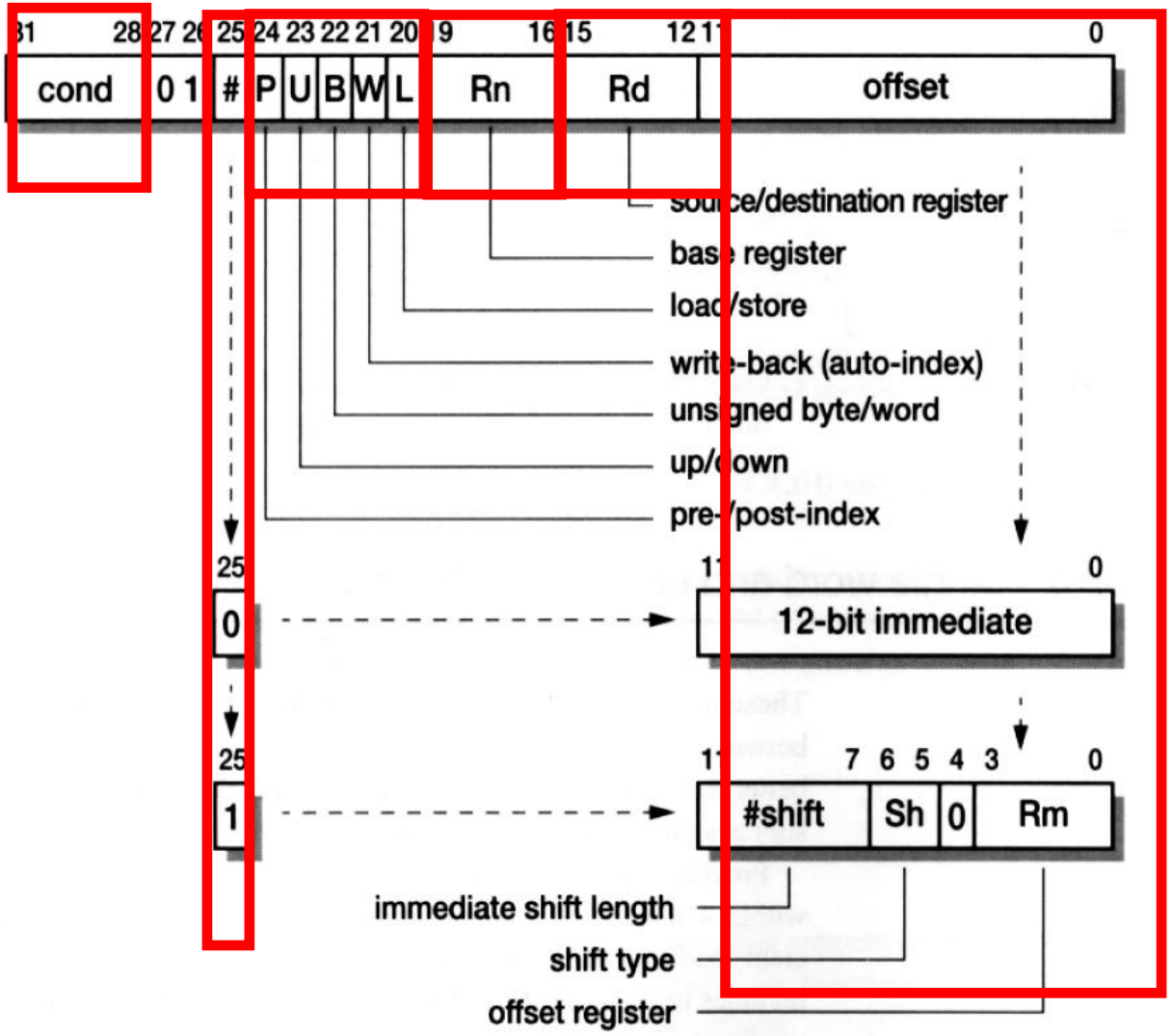
El Registro de Instrucción (IR) – ARM7

- Ejemplo de instrucción genérica de Procesado de Datos:



El Registro de Instrucción (IR) – ARM7

- Ejemplo de instrucción genérica de Transferencia de Datos:



Modos de Direccionamiento

- Para ejecutar una instrucción hay que indicarle donde se encuentran sus operandos, es decir hay que indicarle la dirección donde se encuentran (sus **direcciones efectivas**)
- Para indicar la dirección donde se encuentra el operando se pueden utilizar distintos **modos de direccionamiento**
- Genéricamente se suelen mencionar lo siguientes:
  - 1) Inherente o Implícito
  - 2) Inmediato o Literal
  - 3) Directo o Absoluto (Variante: Directo a registro)
  - 4) Indirecto (presenta 5 variantes)
    - Indirecto con Desplazamiento
    - Indirecto con incremento (pre y post)
    - Indirecto con decremento (pre y post)
    - Indexado
    - Relativo a PC

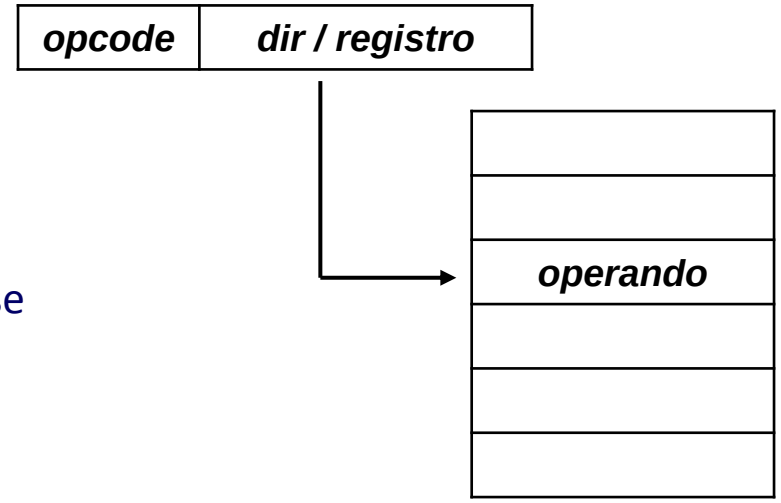
Modos de Direccionamiento

- 1) Inherente o Implícito:
  - El operando se encuentra especificado por el propio código de la instrucción (opcode), no hay operando detrás del opcode

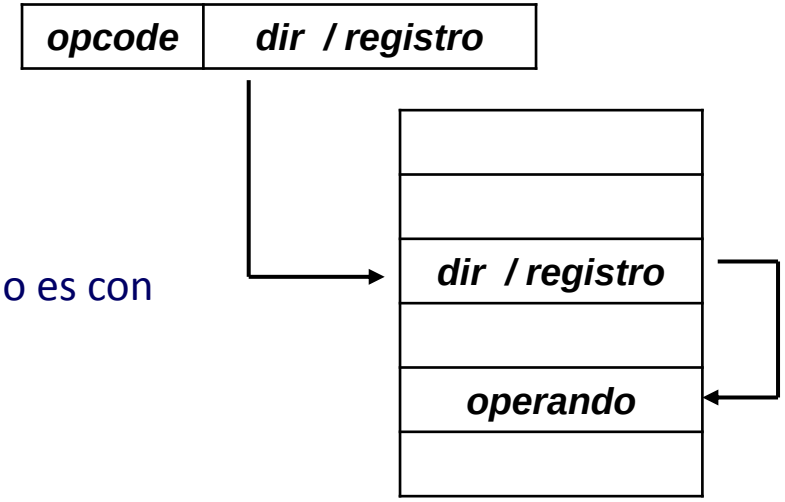
- 2) Inmediato o Literal:
  - El dato se encuentra en la propia instrucción, en los bits siguientes al opcode
    - El dato se suele poner precedido de #
    - Ejemplo: MOVS R2, **#3**
      - R2 $\Leftarrow$ 3

Modos de Direccionamiento

- **3) Directo o Absoluto:**
  - El operando se encuentra en la dirección de memoria indicada en la instrucción (tras el opcode)
    - Ejemplo: LDR R1, **0x00000015**
      - R1 ⇐ (0x00000015)
      - Este ejemplo es ficticio, ya que este modo de direccionamiento no se usa en el ARM7
  - Variante -> **Directo a Registro**, en la que el dato se encuentra en uno de los registros internos del micro
    - Ejemplo: LDR R1, **R2**
      - R1 ⇐ R2
      - Este ejemplo sí es real ya que el modo directo usado en el ARM7

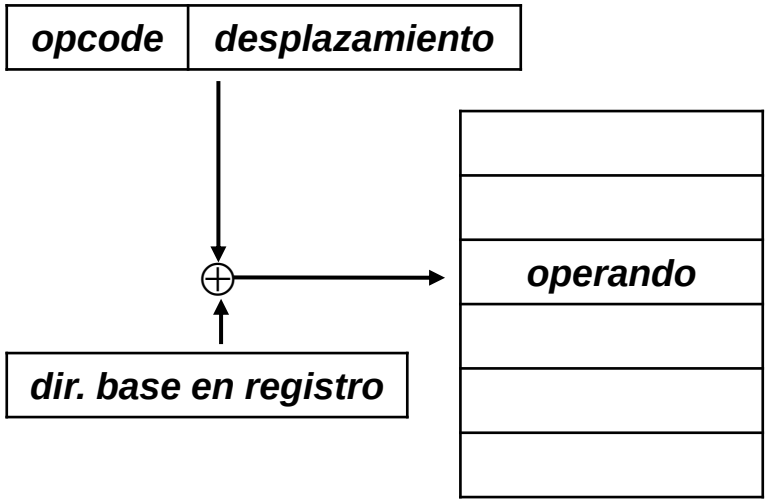


- **4) Indirecto:**
  - El operando se encuentra en la dirección de memoria que se encuentra especificada en un determinado registro o en otra posición de memoria
    - Ejemplo: LDR R1, **[R3]** ⟶ Este ejemplo es con registro
      - R1 ⇐ (R3)
  - Este modo presenta bastantes variantes, que se ven a continuación



Modos de Direccionamiento

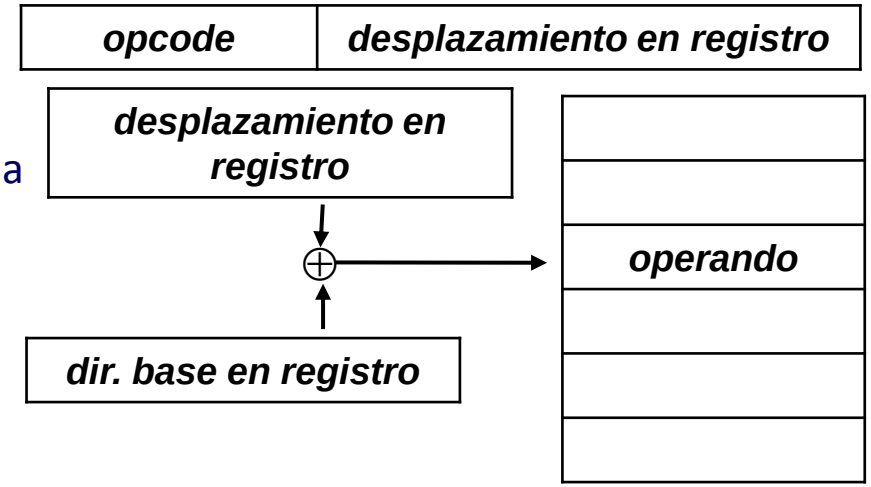
- **Variante -> Indirecto con Desplazamiento:**
  - La dirección efectiva del operando se obtiene al operar el contenido de un registro (que actúa de dirección base), con una determinada cantidad que viene indicada en los bits siguientes al opcode
  - Ejemplo: LDR R1, **[R3, #45]**
    - R1 ⇐ (R3 + 45)
- **Variante -> Indirecto con Incremento**
  - Es un direccionamiento indirecto, pero donde el registro que da la dirección base se modifica:
  - Existe ***pre-incremento***, previamente al cálculo de la dirección efectiva
    - Ejemplo: LDR R1, **[R3, #4]!**
    - R3 ⇐ R3 +4 ;R1 ⇐ (R3);
  - Y ***post-incremento***), posteriormente al cálculo de la dirección efectiva
    - Ejemplo: LDR R1, **[R3], #4**
    - R1 ⇐ (R3); R3 ⇐ R3 +4
- **Variante -> Indirecto con Decremento**
  - Análogo al de incremento pero negativo. Existe ***pre-decremento***
    - Ejemplo: LDR R1, **[R3, #-4]**
    - R3 ⇐ R3 -4 ;R1 ⇐ (R3);
  - Y ***post-decremento***
    - Ejemplo: LDR R1, **[R3],#-4**
    - R1 ⇐ (R3); R3 ⇐ R3 -4



La nomenclatura sería como el indirecto con desplazamiento, por lo que para diferenciarlo se pone un ! al final

Modos de Direccionamiento

- **Variante -> Indexado:**
  - Idéntico al indirecto con desplazamiento, pero en el que la cantidad que se le suma a la dirección base, está en otro registro
  - Ejemplo: LDR R1, **[R3, R8]**
    - R1 ⇐ (R3 + R8)
  - Puede llamarse como "indexado con desplazamiento"



- **Variante -> Relativo a PC:**
  - Se trata de un direccionamiento indirecto o indexado, donde la dirección base se encuentra en el PC
  - Es el modo de direccionamiento utilizado por determinadas llamadas a subrutinas locales
  - Ejemplo: LDR R1, **[PC, #300]**
    - R1 ⇐ (PC + 300)