Tema 2

Arquitectura y Modelo de Programación del Cortex M3

Indice

2.0 Introduccion

- Cortex-M3 es un procesador de 32 bits diseñado (no fabricado) por ARM y basado en la arquitectura ARMv7
- Pertenece al perfil **M** de los procesadores con esa arquitectura **(ARMv7-M)**:
  - Procesadores para aplicaciones de bajo-costo en los que la eficiencia de procesado es importante, y el coste, consume, baja latencia de interrupción y facilidad de uso son críticos
  - Especialmente indicados para aplicaciones de control industrial, incluidos sistemas de control en tiempo-real

2.1 Arquitectura del Cortex-M3

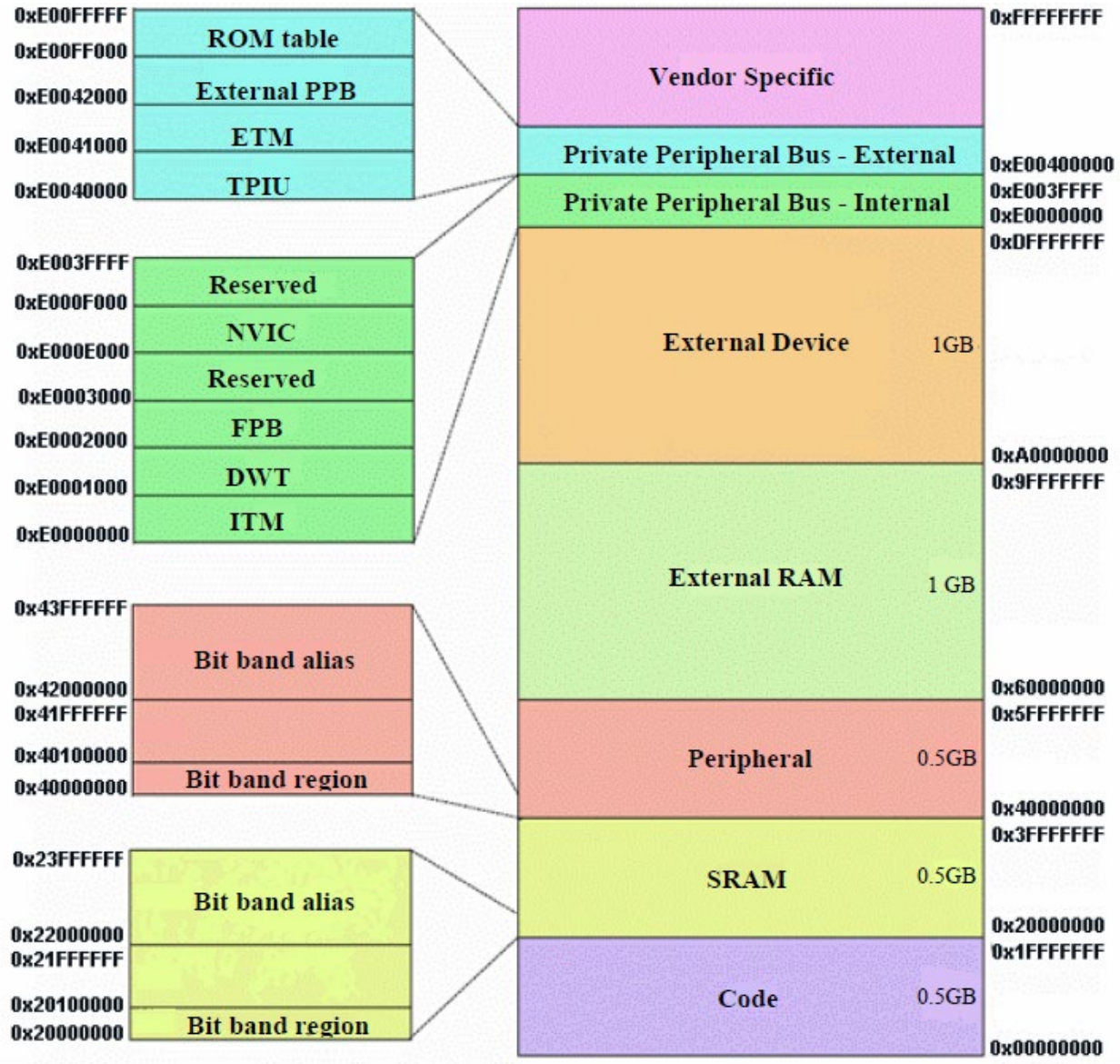
Características de su arquitectura

- Microcontrolador de 32 bits
- Arquitectura Harvard: 2 conjuntos diferentes de buses, uno para codigo y otro para datos y periféricos.
- Pipelining de 3 etapas con especulacion de salto
- Multiplicadores de 32 bits de ciclo único
- Unidades de division con y sin signo operando entre 2 y 12 ciclos
- 4 Gigabytes de mapa de memoria pre-configurados
- Mapa de memoria gestionable por la MPU en regiones desde 32bytes a 4Gbytes.
- Acceso a memoria en un unico ciclo, incluso a datos no alineados
- 2 zonas de 1Mbit para acceso a datos de 1 bit
- 1.25 DMIPS/MHz (millones de instrucciones enteras por segundo)

2.1 Arquitectura del Cortex-M3

- El Cortex-M3 tiene un mapa de memoria predefinido de 4Gbytes
  - Perifericos incorporados al chip, NVIC, unidades de depuracion, … **se pueden acceder mediante una simple instrucción de acceso a memoria**
- Espacio de memoria esta dividido
  - El diseño del Cortex-M3 tiene una infraestructura de bus interno optimizado para este uso de memoria
- La Portabilidad esta asegurada tambien entre fabricantes diferentes
  - Aunque las partes no-definidas en el mapa de memoria pueden cambiar entre dispositivos diferentes:
    por ejemplo, el LPC1850 tiene mas memoria externa mapeada que el LPC1788

2.1 Arquitectura del Cortex-M3



2.1 Arquitectura del Cortex-M3

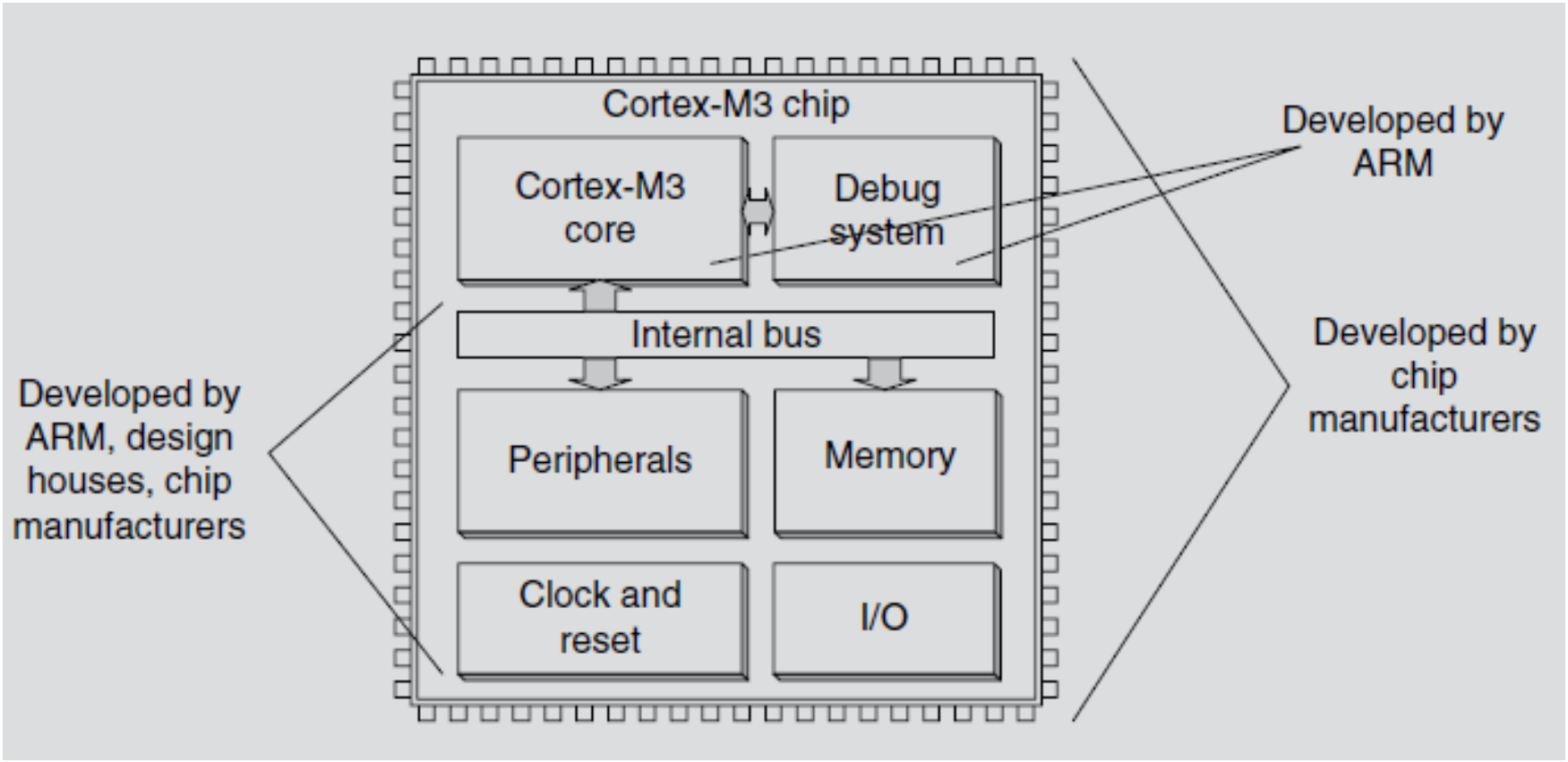
- Endiannes
  - En Cortex-M3, el modo endian se establece cuando el procesador sale del reset
    - No se puede cambiar tras él (no hay conmutacion de endian dinámica)
  - Algunos espacios de memoria tienen un acceso little endian fijo
  - El esquema Big endian se concoe como byte-invariant big endian, tambien llamado BE-8 (soportado en la arquitectura v6 y v7 de ARM)

Application Interrupt and Reset Control Register 

[15]	ENDIANESS	RO	Data endianness bit: 0 = Little-endian.

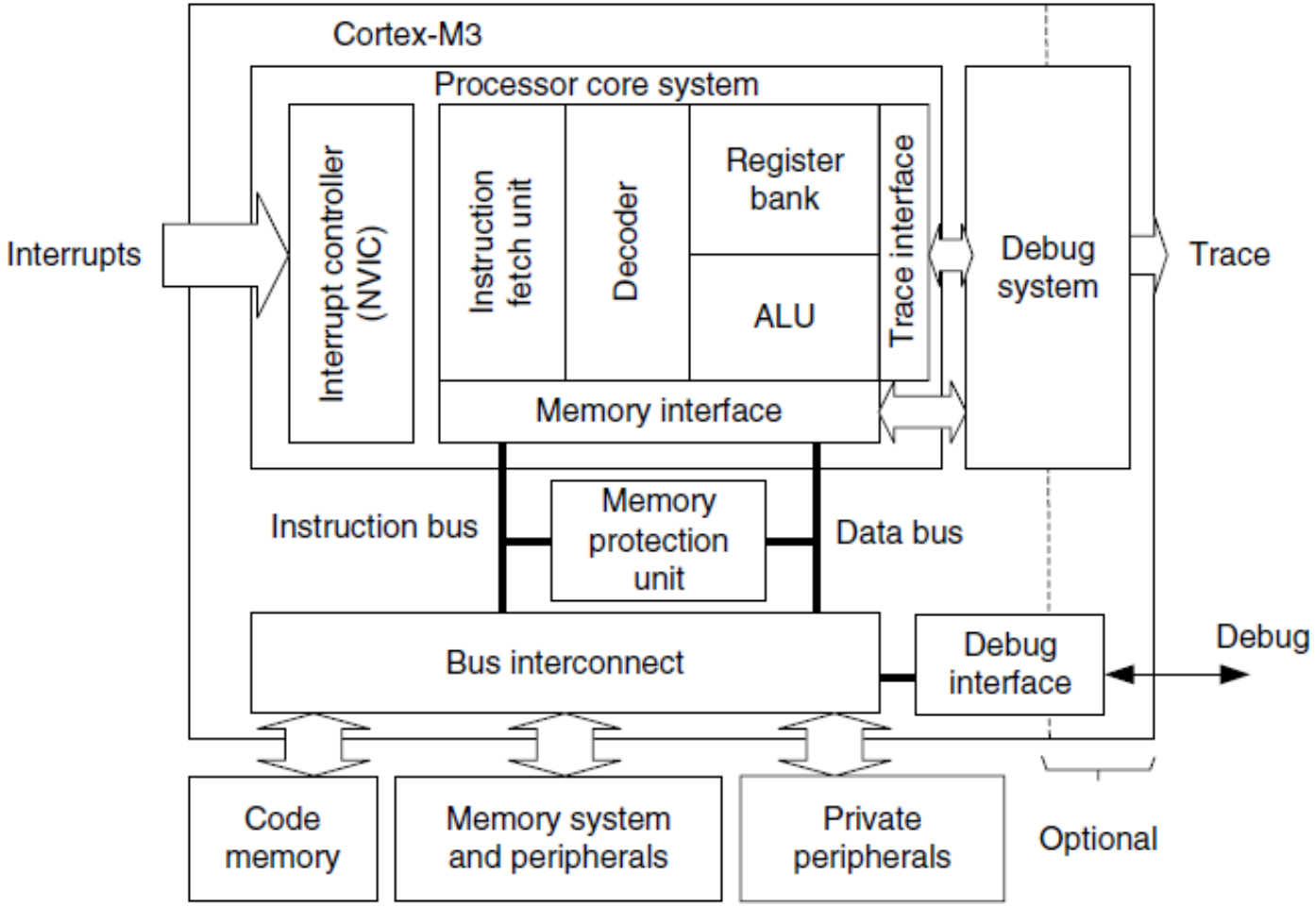
2.1 Arquitectura del Cortex-M3

ARM Cortex-M3

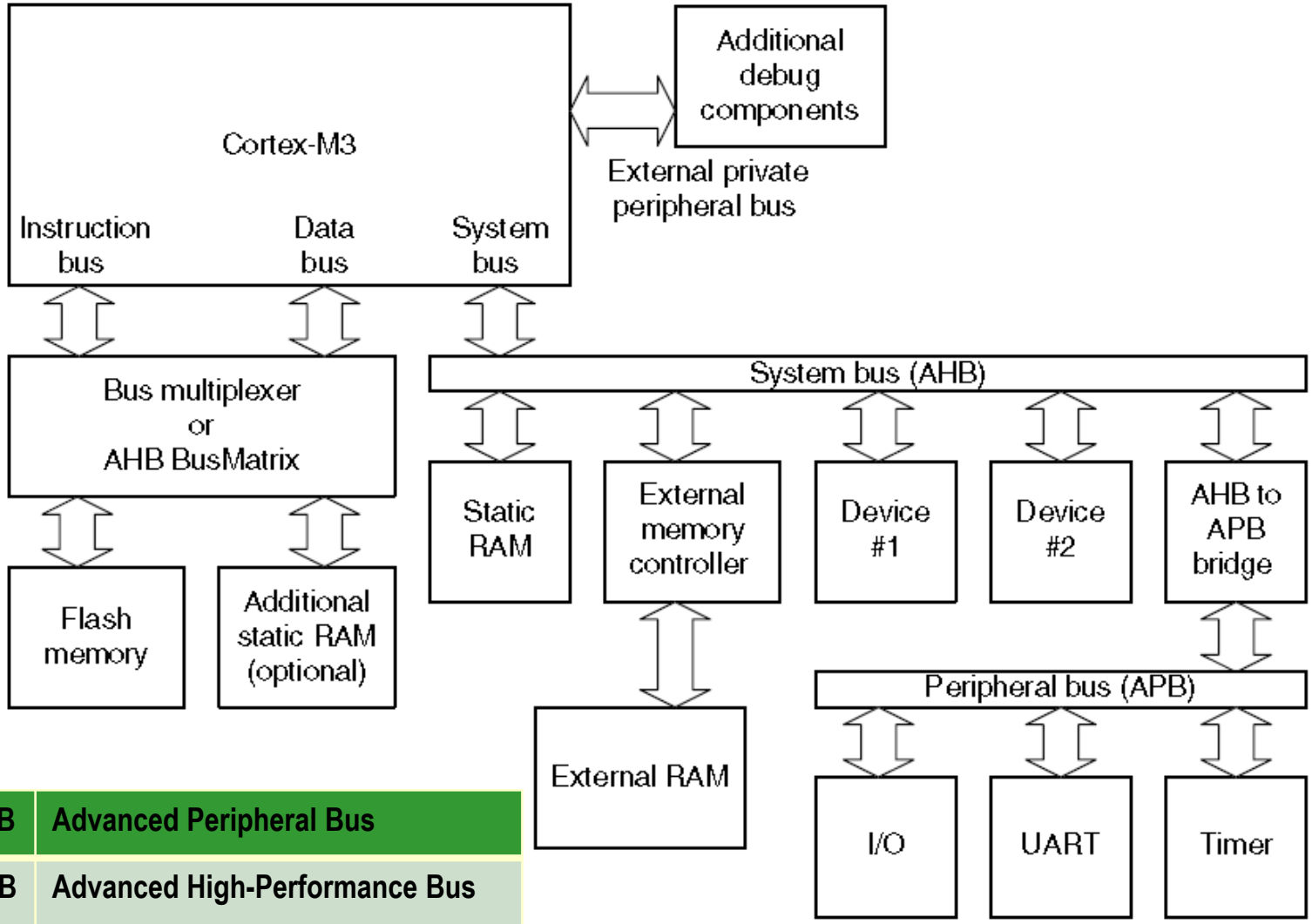


2.1 Arquitectura del Cortex-M3

- Modelo de programación



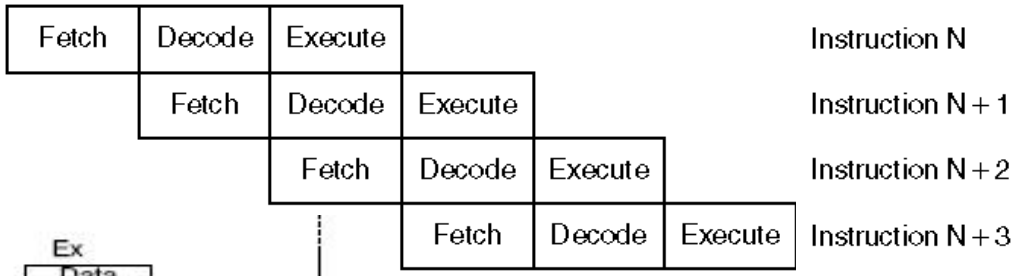
2.1 Arquitectura del Cortex-M3

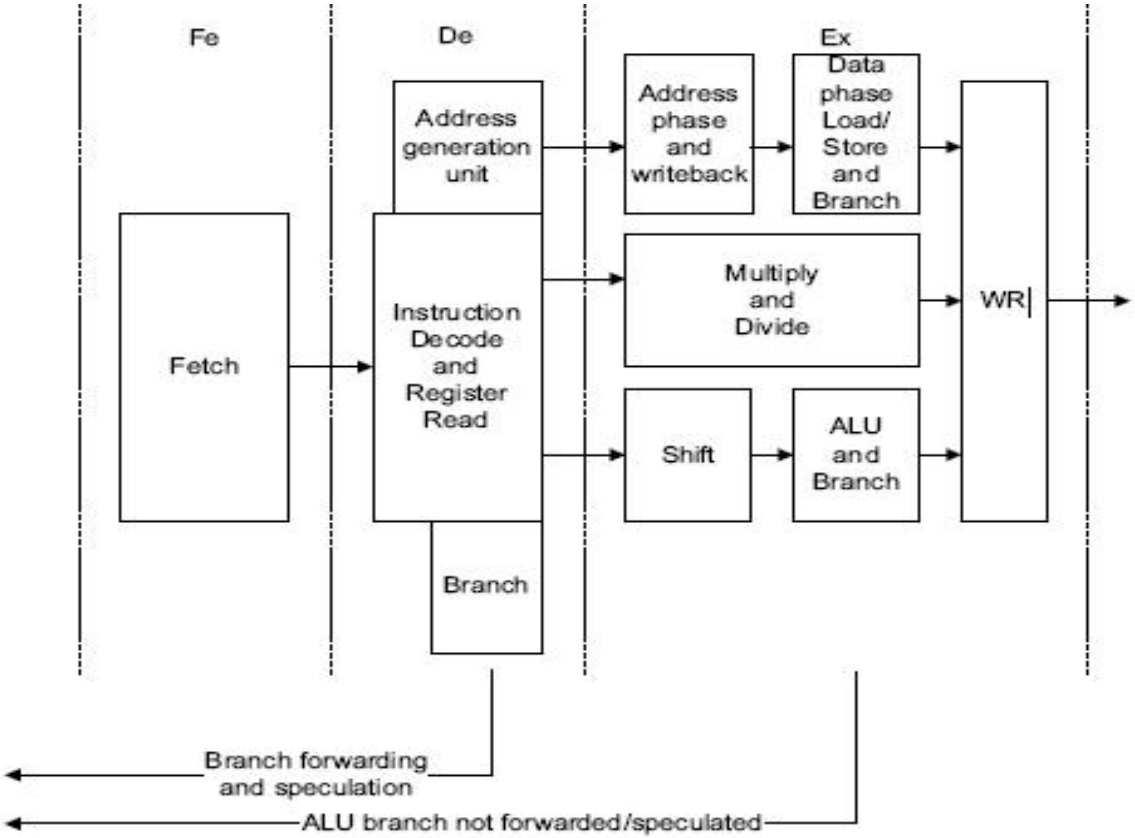


APB	**Advanced Peripheral Bus**
AHB	**Advanced High-Performance Bus**

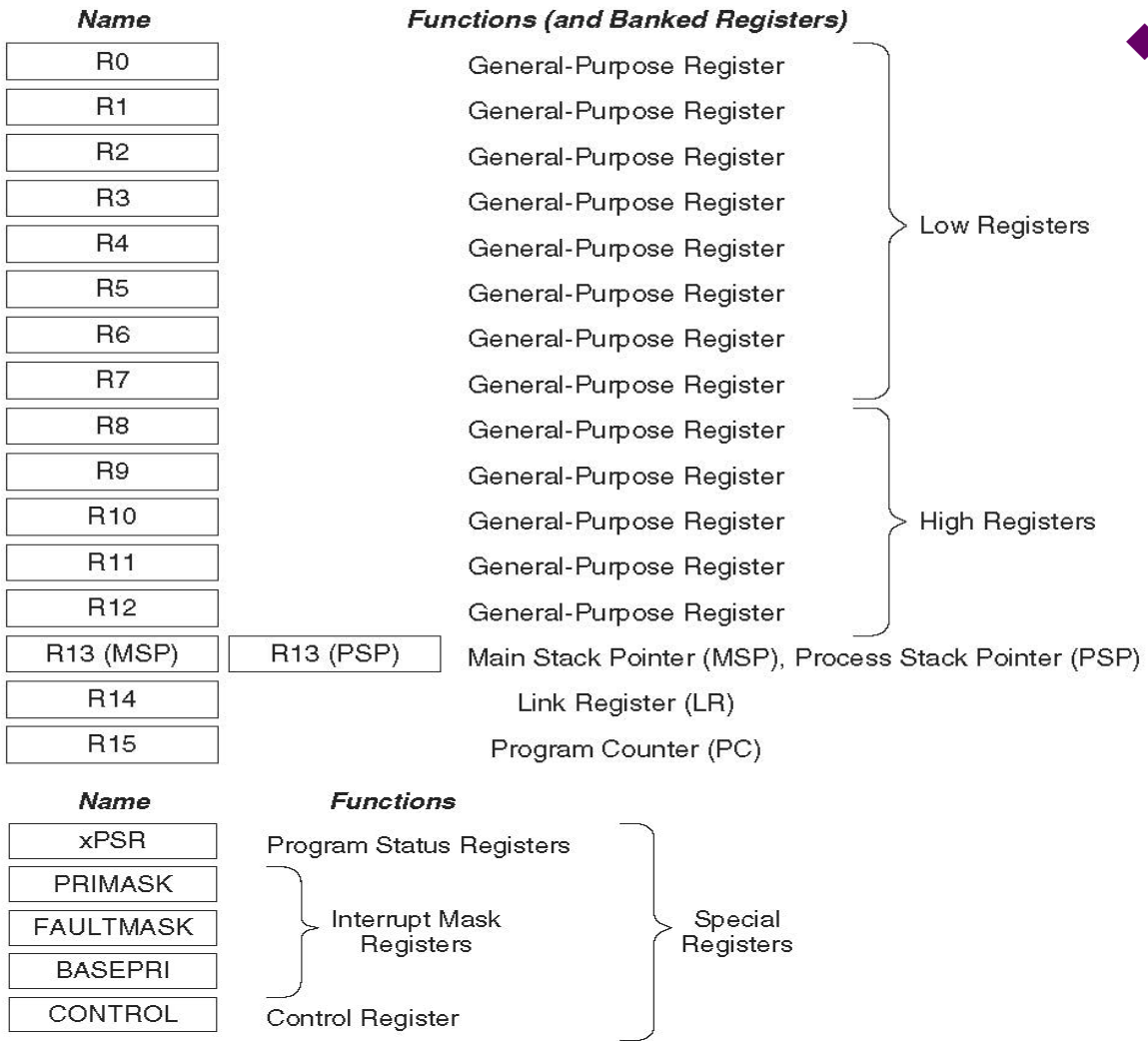
2.1 Arquitectura del Cortex-M3

- El Cortex M3 usa una segmentación de tres etapas: **fetch, decode and execution**
- ¿Que ocurre en cada etapa?





2.2 Modelo de programación: Registros



- **Tipos de datos soportados por el procesador**
  - 32-bit ⟶ words
  - 16-bit ⟶ halfwords
  - 8-bit ⟶ bytes

2.2 Modelo de programación: Registros

- Registros
  - El procesasdor Cortex-M3 posee los registros internos desde el R0 al R15
  - **R0 to R12: Registros de Proposito General (General-Purpose Registers,GPRs)**
  - **R13 es el Puntero de Pila (Stack Pointer, SP):** Realmente son dos registros, pero en cada momento solo existe un R13 visible
  - **R14 es el Registro de Enlace (Link Register, LR):** utilizado en llamadas a subrutinas (o funciones) y manejadores de excepcion (exception handlers)
  - **R15 es el Registro Contador de Programa (Program Counter, PC)**
- Registros de Propósito General (GPRs): Bajos y Altos
  - Del R0 al R12 son GPRs de 32 bit para operaciones de datos

2.2 Modelo de programación: Registros

- Registro R13: **Punteros de Pila (Stack Pointers, SPs)**
  - Cortex-M3 contiene dos registros de pila, llamados R13
    - Main Stack Pointer (MSP)
    - Process Stack Pointer (PSP)
  - Pese a ser dos registros físicos distintos, unicamente se encuentra visible uno de ellos en cada momento, dependiendo del nivel de privilegios de procesador (se verá más adelante)
  - Cuando usas el nombre de registro R13, accedes al registro que en ese momento se encuentre visible (el otro estará "oculto")
  - **Los dos bits mas bajos del registro de pila siempre valen 0, lo que significa que siempre se encuentran alineados al tamaño de palabra**

2.2 Modelo de programación: Registros

- Registro R14: **Registro de enlace (Link Register, LR)**
  - LR se utiliza para almacenar la direccion de retorno (que se cargará en el PC) cuando se invoca una subrutina o funcion
  - LR debe ser tratado manualmente para desarrollar de forma correcta subrutinas anidadas, cuando se programe en lenguaje ensamblador.

2.2 Modelo de programación: Registros

- Registro R15: **Contador de Programa (Program Counter, PC)**
  - Como sabemos, mantiene la direccion de la memoria de programa donde se encuentra la siguiente instruccion a ejecutar.
  - En Cortex-M3, el valor del registro marca la localizacion de la dirección mas 4:

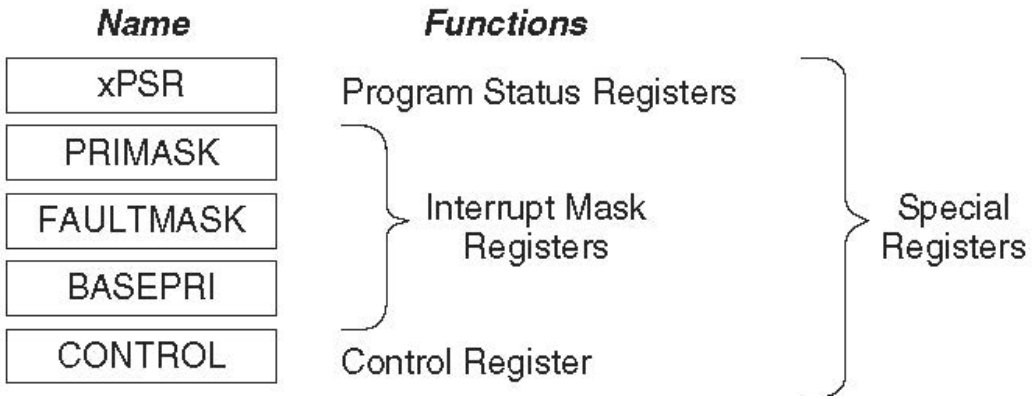
```
0x1000:    MOV R0, PC          ; R0 = 0x1004
```

  - Una instruccion que escriba en el registro PC provocará un salto en la ejecución del programa
  - Debido a que una instrucción puede estar alineada a media palabra (half word), el bit menos significativo del registro PC (bit 0) siempre vale 0.

2.2 Modelo de programación: Registros

Registros Especiales:

- Registros de Estado (Program Status Registers): **APSR, IPSR, EPSR**
- Registros de Mascara de Excepcion (Exception Mask Registers): **PRIMASK, FAULTMASK, BASEPRI**
- Registro de Control: **CONTROL**



- **xPSR** Provide ALU flags (zero flag, carry flag), execution status, and current executing exception number
- **PRIMASK** Disable all exceptions except the nonmaskable interrupt (NMI) and HardFault Effectively changes the priority level to 0 (highest programmable level)
- **FAULTMASK** Disable all exceptions except the NMI. Changes the priority level to -1
- **BASEPRI** Disable all exceptions of specific priority or lower priority level
- **CONTROL** Define privileged status and stack pointer selection

2.2 Modelo de programación: Registros

- Registros Especiales:
  - Tiene funciones especiales y solo puede ser accedidos por instrucciones especiales: No se puede utilizar con instrucciones normales de procesamiento de datos
- Registros de Estado de Programa (Program Status Registers, **PSRs)**
  - Divididos en tres registros de Estado:
    - **Application PSR (APSR)**
    - **Interrupt PSR (IPSR)**
    - **Execution PSR (EPSR)**

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

2.2 Modelo de programación: Registros

- Registros de Estado de Programa (PSRs): **APSR, IPSR**

Application Program Status Register bit assignments

Field	Name	Definition
[31]	N	Negative or less than flag: 1 = result negative or less than 0 = result positive or greater than.
[30]	Z	Zero flag: 1 = result of 0 0 = nonzero result.
[29]	C	Carry/borrow flag: 1 = carry or borrow 0 = no carry or borrow.
[28]	V	Overflow flag: 1 = overflow 0 = no overflow.
[27]	Q	Sticky saturation flag.
[26:0]	-	Reserved.

Interrupt Program Status Register bit assignments

Field	Name	Definition
[31:9]	-	Reserved.
[8:0]	ISR NUMBER	Number of pre-empted exception. Base level = 0 NMI = 2 SVCall = 11 INTISR[0] = 16 INTISR[1] = 17 . . . INTISR[15] = 31 . . . INTISR[239] = 255

2.3 Instrucciones

Lenguaje Ensamblador: Sintaxis Básica

- Para la programación del Cortex-M3 mediante lenguaje ensamblador, el formato mostrado a continuación es el usado normalmente:

```
etiqueta:    opcode operando1, operando2,... ; Comentarios
```

- El numero de operandos depende del tipo de instruccion.
- El tamaño de la instruccion en Cortex-M3 puede ser de 16 o 32 bits
  - Por ejemplo:

```
0x1000:    LDR R0,[R1]          ; instruccion de 16 bits (ocupa direccion 0x1000-0x1001)
0x1002:    RBIT.W R0            ; instruccion  de 32bits (ocupa direccion 0x1002-0x1005)
```

2.4 Modos de Direccionamiento

- Cortex-M3 suporta los siguientes modos de direccionamiento:
  - **Inmediato**
  - **Registro** (Directo a Registro)
  - **Indirecto**
  - **Offset** o Base+Offset (Indirecto+Offset)
  - **Indexado** o Base+Indice+Offset
  - **Pre-indexado**
  - **Post-indexado**
  - **relativo a PC**

2.4 Modos de Direccionamiento

- Inmediato
  - El operando es un número
  - Sintaxis en ensamblador: *#numero*
  - Sólo se puede utilizar para un operando fuente
  - Válido en todo tipo de instrucciones
  - Por ejemplo:

```
ADD R0, #0x12        ; R0 = R0+0x12 (hexadecimal)
MOV R1, #'A'         ; Set R1 = caracter ASCII 'A'
```

2.4 Modos de Direccionamiento

- Registro (Directo a Registro)
  - El dato está almacenado en un registro (GPRs o SRs): *EA = rx*
  - La sintaxis en ensamblador es: *rx*
  - Válido en todo tipo de instrucciones
  - Por ejemplo:

```
ADD  R0, R1                    ; R0 = R0+R1
```

- Indirecto
  - El dato está en la dirección de la memoria apuntada por el contenido de un registro :

    EA = content of rx
  - La sintaxis en ensamblador es: *[rx]*
  - Sólo aplicable en instruciones de transferencia de datos
  - Por ejemplo:

```
LDR R0,[R3]          ; R0 = contenido de la memoria apuntado por R3
```

2.4 Modos de Direccionamiento

- Offset o Base+Offset (Indirecto+Offset)
  - El dato esta en la direccion de memoria obtenida tras la siguiente operacion:

    EA = contenido de rx + offset
  - La sintaxis en ensamblador es: *[rx,#offset]*
  - El valor del registro no se modifica
  - Sólo aplicable en instruciones de transferencia de datos
  - Por ejemplo:

```
LDR R0,[SP,#24]      ;R0 = contenido de la memoria apuntado por: el contenido de SP+24
```

2.4 Modos de Direccionamiento

- Indexado o Base+Indice(desplazado)
  - El dato se encuentre en la direccion de memoria obtenida tras la siguiente operacion:
    E*A = contenido del registro rx + contenido del registro rIndice(desplazado)*
  - La sintaxis es: *[rx,rIndice,LSL#n] ; n=1,2,3 para desplazar rIndice antes que se obtenga EA*
  - Ambos registros permanecen sin modificacion
  - Sólo aplicable en instruciones de transferencia de datos
  - Por ejemplo:

```
LDR R0,[R1,R2,LSL#2]   ; R0 = contenido de la memoria apuntado por R1+R2*4
```

2.4 Modos de Direccionamiento

- Pre-indexado
  - El dato se encuentra en la misma direccion que el modo Indirecto+Offset:

    EA = contenido de rx+offset

  - La sintaxis en ensamblador es: *[rx,#offset]!*
  - El valor del registro se modifica ANTES de que se produzca el acceso
  - Sólo aplicable en instruciones de transferencia de datos
  - Por ejemplo:

```
LDR R0,[R2,#2] !      ; R2 = R2 + 2
                      ; R0 = contenido de la memoria apuntado por R2+2
```

2.4 Modos de Direccionamiento

- Post-indexado
  - El dato se encuentra en la misma ubicacion que el modo Indirecto: *EA = content of rx*
  - La sintaxis en ensamblador es: *[rx],#offset*
  - El valor del registro se altera DESPUES de realizado el acceso
  - Sólo aplicable en instruciones de transferencia de datos
  - Por ejemplo:

```
LDR R0,[R2],#2        ; R0 = contenido de la memoria apuntado por R2
                      ; R2 = R2 + 2
```

2.4 Modos de Direccionamiento

- Relativo a PC
  - El dato se encuentre en la direccion de memoria obtenida tras la siguiente operacion:

    EA = contenido de PC + offset
  - Solo se utiliza con ciertas instrucciones especificas (Saltos y carga de direcciones)
  - La sintaxis en ensamblador es: *etiqueta*
  - Por ejemplo:

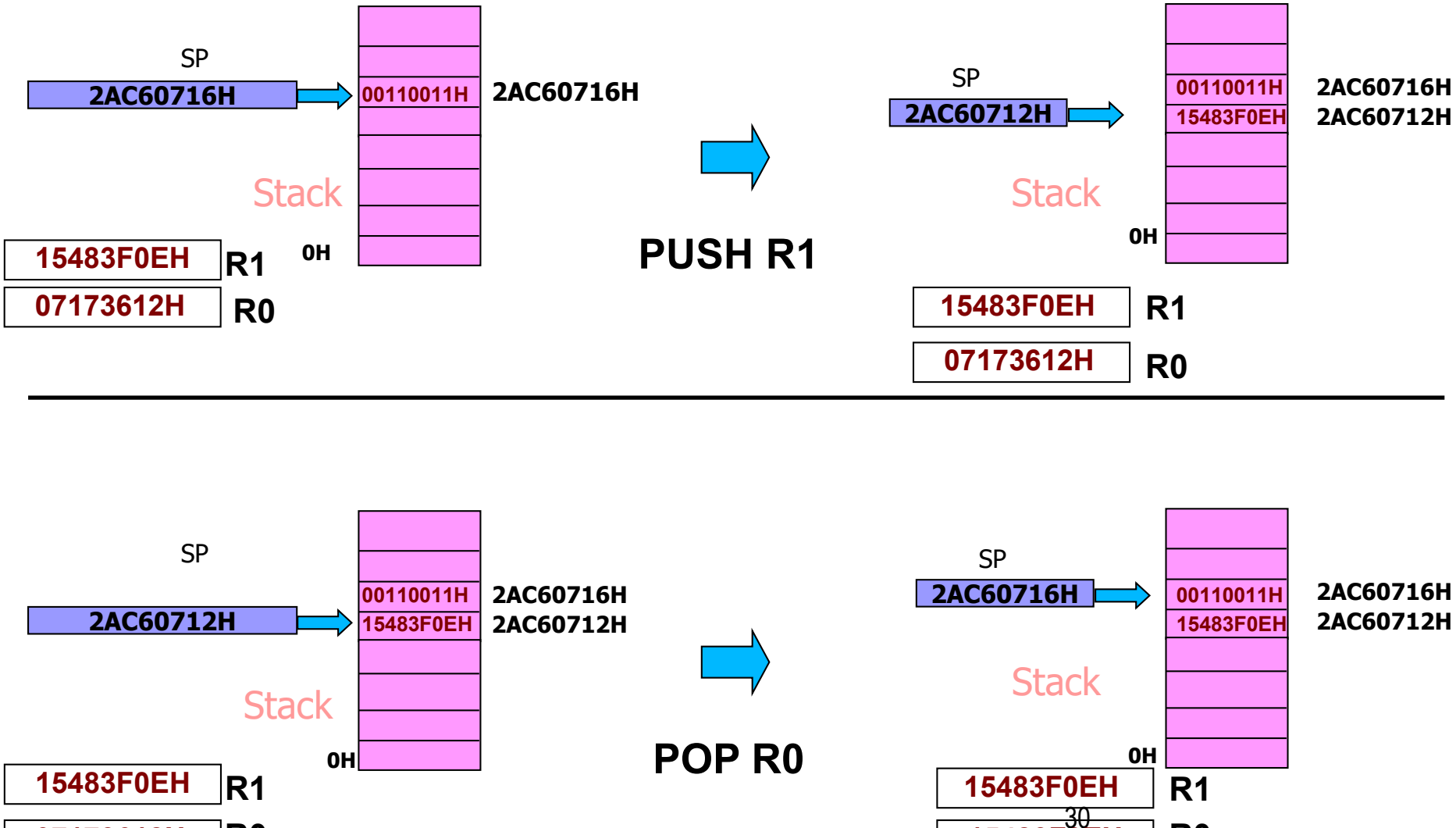
```
ADR R1,etiqueta   ;  R1 = PC+etiqueta
```

2.5 Tipos de Instrucciones

- **Lista de Instrucciones:**
  - Instrucciones de Transferencia(en tamaño byte,half word, word y double)
    - Dato inmediato a registro
    - De registro a registro
    - Entre registro especial y registro
    - Entre memoria y registro (Load y Store, LDR y STR respectivamente)
    - Entre registro y la pila (PUSH y POP)
  - Instrucciones de procesados de datos
    - Aritméticas
    - Logicas y de manipulacion de bits
    - Rotacion y desplazamiento
  - Instrucciones de salto
    - Condicional
    - Incondicional
    - Llamada a subrutina
  - Otras instrucciones de 32 Bits
- **Todas incluidas en el Conjunto de Instrucciones(Instructions Set)**

2.6 Pila (Stack)

- Operaciones basicas de la pila



2.6 Pila (Stack)

Operaciones basicas de la Pila

- Los datos almacenados en registros se guardan en la memoria de la pila mediante una operacion PUSH…
- …y puede ser restablecido a registros mas tarde mediante una operacion POP
- El valor del registro SP se ajusta automaticamente en las operaciones PUSH y POP, por lo que multiples PUSH consecutivos no borrarán los datos previamente insertados.
- Importante el orden de PUSH y POP: El orden de **POP debe ser el inverso al orden de PUSH**
- Operaciones donde interviene la Pila, como **Cambio de contexto y Paso de Parámetros**
  - Se simplifican, gracias a las instrucciones PUSH y POP con multiple carga y almacenamiento
  - The procesador automaticamente invierte en POP el orden de la lista de registros de PUSH

2.6 Pila (Stack)

Implementacion del Stack

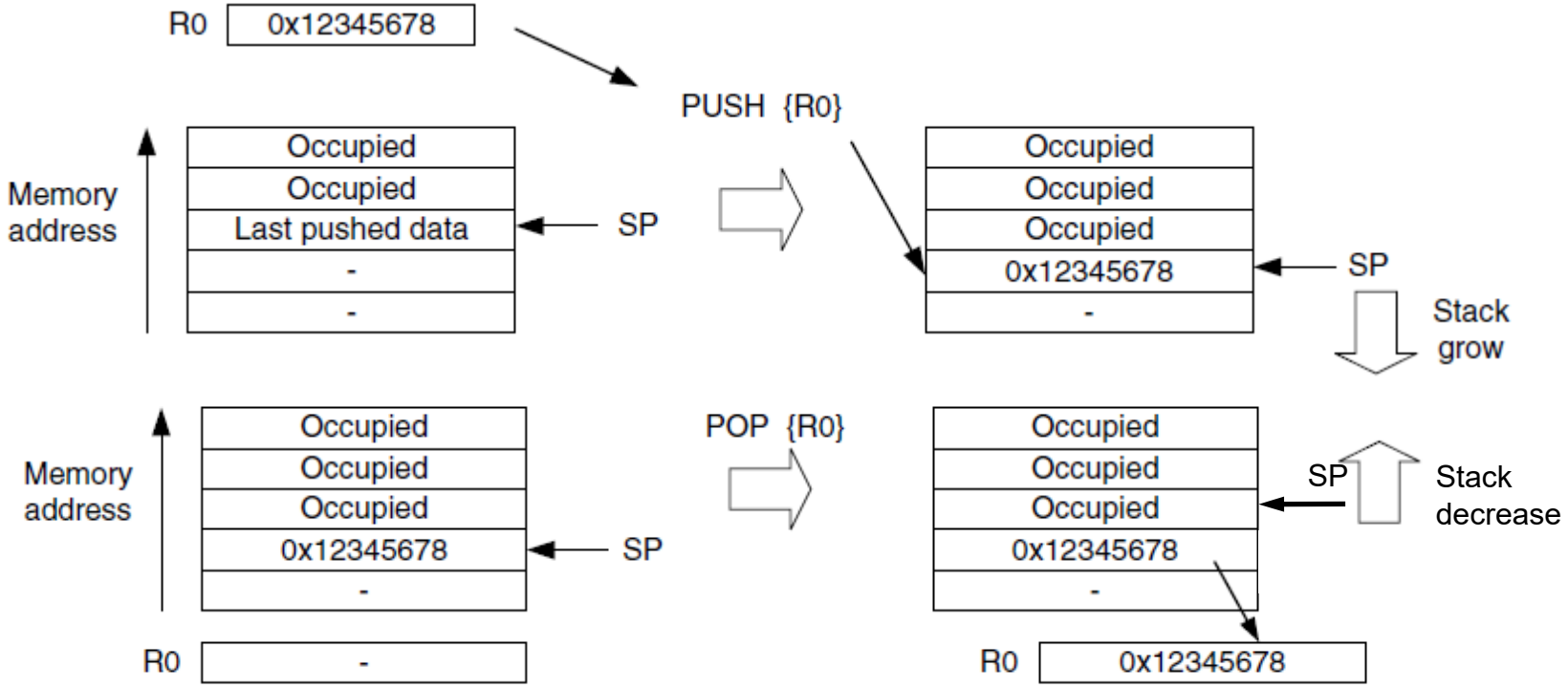
- Las operaciones PUSH y POP solo se pueden hacer sobre registros
- Cada operacion PUSH/POP **transfiere 4 bytes of datos** (la word completa del registro)
- Ademas, el valor del registro SP se decrementa/incrementa en 4 en cada operacion PUSH/POP (respectivamente) o en un multiplo de 4 si se apilan mas de 1 registro

Uso de la pila en Manejo de Excepciones (ISR)

- Cuando se entra a una ISR, **un numero de registros se almacenan en la pila automaticamente**, y R13 se usará como registro SP para este proceso
- De foma analoga, **los registros almacenados se restableceran automáticamente cuando se salga de la ISR**, y el registro SP tambien será adjustado.

2.6 Pila (Stack)

- El Cortex-M3 utiliza un modelo de operacion con la pila full-descending
  - **SP apunta al ultimo dato introducido en el** stack
  - SP en la operacion PUSH se **decrementa antes de almacenar en la pila, y se incrementa en la operacion POP despues de sacar el dato de la pila**



2.6 Pila (Stack)

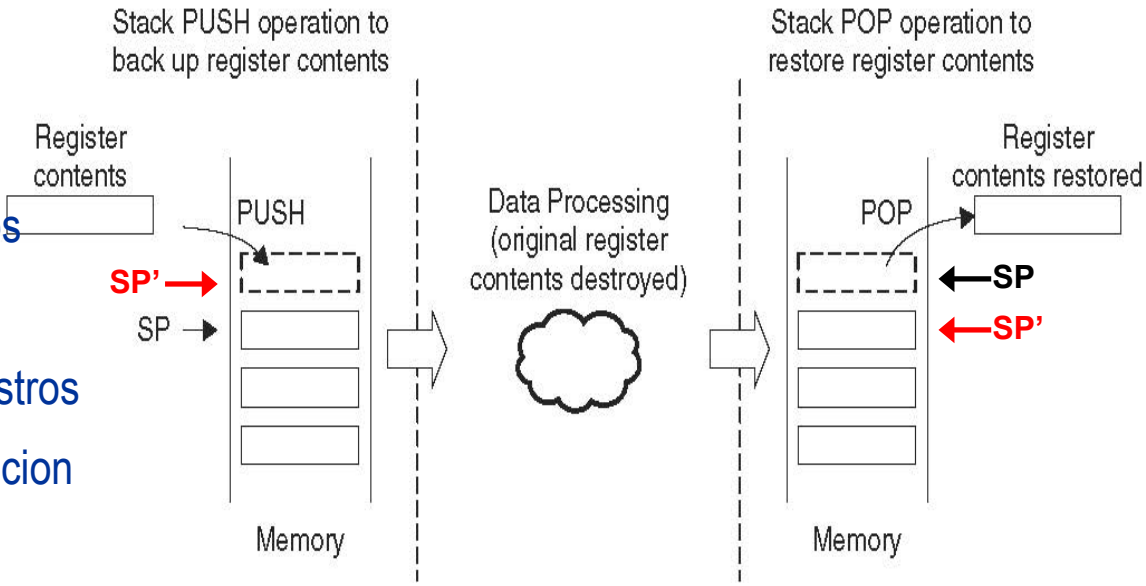
- **Guardando en la pila un solo registro:**

  PUSH {R0} ; R13 = R13-4, y luego Mem[R13] = R0

  POP {R0} ; R0 = Mem[R13], y luego R13 = R13+4

- **Llamada a subrutina:**

```
subroutine_1
   PUSH {R0-R7, R14} ; Guarda registros
   …                 ; Procesado
   POP {R0-R7, R14}  ; Restablece registros
   BX R14            ; Retorna a la funcion
```



2.6 Pila (Stack)

Ejemplo de uso de pila

- **Llamada a subrutina**

Main program

```
    ...
; R0   X, R1   Y, R2   Z
BL    function1
```

Subroutine

```
function1
    PUSH    {R0} ; store R0 to stack & adjust SP
    PUSH    {R1} ; store R1 to stack & adjust SP
    PUSH    {R2} ; store R2 to stack & adjust SP
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R2} ; restore R2 and SP re adjusted
    POP     {R1} ; restore R1 and SP re adjusted
    POP     {R0} ; restore R0 and SP re adjusted
    BX      LR   ; Return
```

```
; Back to main program
; R0   X, R1   Y, R2   Z
... ; next instructions
```

2.6 Pila (Stack)

Ejemplos de uso de pila

- Llamada a rutina con instrucciones PUSH/POP con multiple carga y restablecimiento

Main program

```
    ...
; R0   X, R1   Y, R2   Z
BL    function 1
```

Subroutine

```
function 1
    PUSH    {R0 R2} ; Store R0, R1, R2 to stack
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R0 R2} ; restore R0, R1, R2
    BX      LR   ; Return
```

```
; Back to main program
; R0   X, R1   Y, R2   Z
... ; next instructions
```

- Para subrutinas (functiones) anidadas LR tambien debe ser metido en la pila

Main program

```
    ...
; R0   X, R1   Y, R2   Z
BL    function 1
```

Subroutine

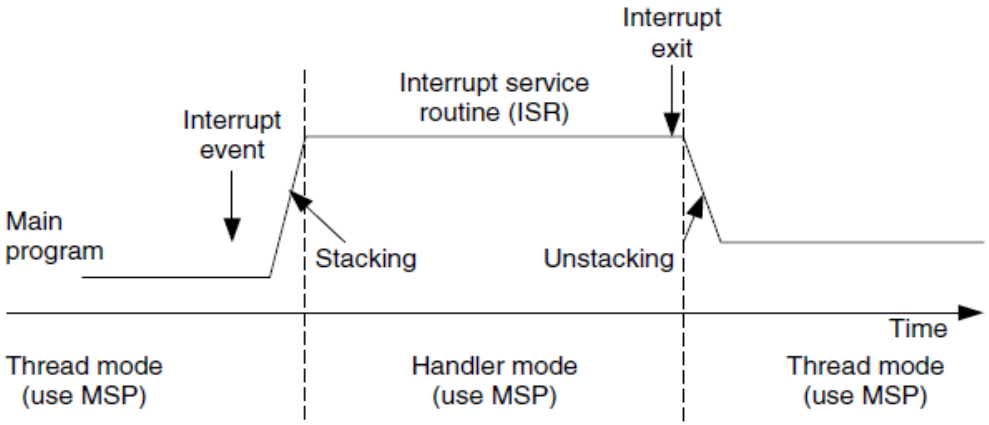
```
function 1
    PUSH    {R0 R2, LR} ; Save registers
                ; including link register
    ... ; Executing task (R0, R1 and R2
        ; could be changed)
    POP     {R0 R2, PC} ; Restore registers and
                ; return
```

```
; Back to main program
; R0   X, R1   Y, R2   Z
... ; next instructions
```

2.6 Pila (Stack)

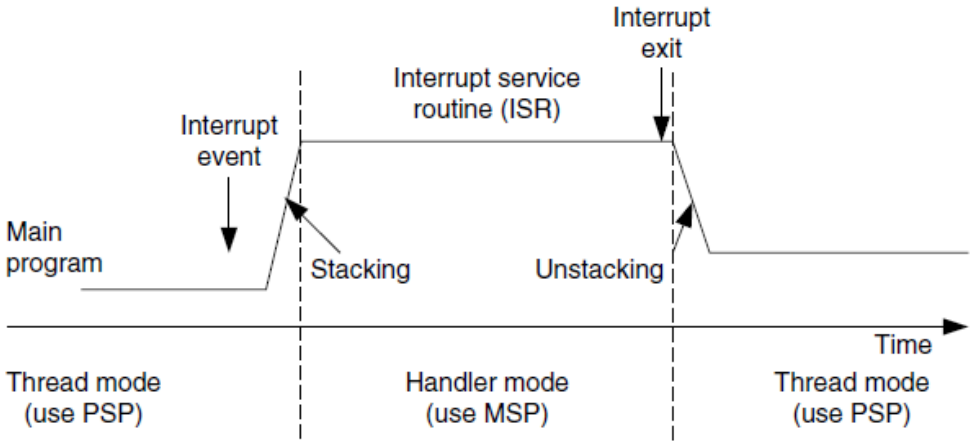
El modelo de Dos-Pilas

- Cortex-M3 tiene dos SPs: el MSP y el PSP
- El registro SP que se usa en cada momento se controla con el registro CONTROL[1]
- Si CONTROL[1]=0, el MSP se usa tanto en modo Thread como en Handler:
  - El programa principal y las rutinas de atencion a la excepcion comparte la misma region de pila
  - Esta es la configuracion por defecto tras un encendido



2.6 Pila (Stack)

- Si CONTROL[1]=1, el PSP se usa para el modo Thread:
  - El programa principal y las rutinas de atencion a la excepcion pueden tener regiones de memoria de pila distintas
    - El manejo de la pila automatico utilizará el PSP cuando se salte a una rutina de atencion a la excepcion
    - Las operaciones con el stack dento de dicha rutina utilizaran el MSP
  - Es posible **desarrollar operaciones de lectura/escritura directamente al MSP y al PSP (sin usar el nombre R13 o SP)**, evitando la confucion del registro R13 al que te refieres
  - Si el procesador se encuentra en nivel privilegiado, **se puede acceder a los valores de los registros MSP y PSP**



2.7 Reloj de CPU

- El procesador realiza todas sus funciones sincronizado con una señar de reloj
- ¿A que frecuencia trabaja el procesador?
- El LPC1768 incluye tres fuentes de oscilador independentes:
  - Main Oscillator
  - Internal RC Oscillator
  - RTC Oscillator.
- Tras un reset, el LPC1768 opera usando el Internal RC Oscillator, mientras no se cambie la configuracion por software.

2.B Introduction

- Relojes del LPC1768.

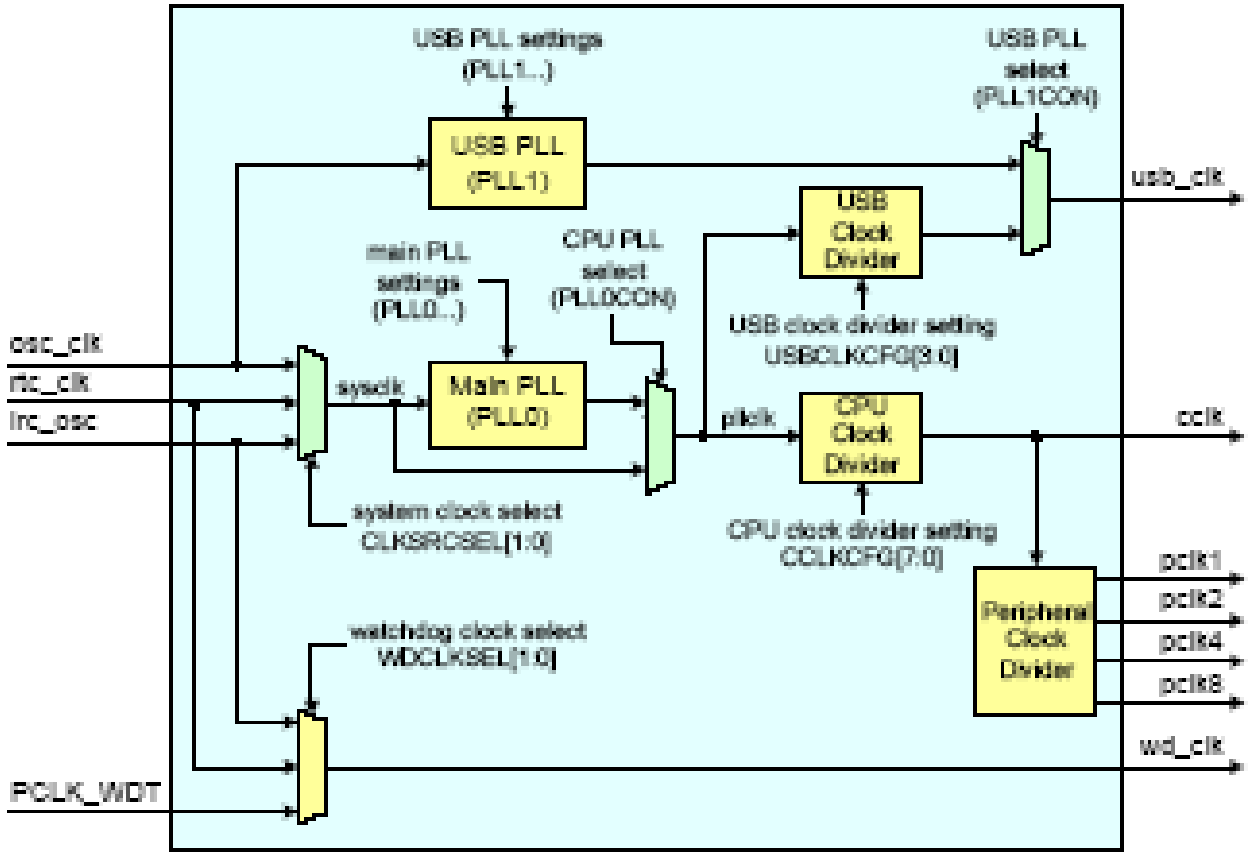


Fig 7. Clock generation for the LPC17xx

2.7 Reloj de CPU

- Oscilador Interno RC (IRC)
  - Se puede utilizar para la funcion de watchdog timer, o para sincronizar la ejecución de instrucciones en la CPU.
  - El valor de frecuencia nomina del IRC es 4 Mhz.
- El main Oscillator
  - Se usa para sincronizar la ejecución de instrucciones en la CPU (aplicación mas común).
  - Trabaja a una frecuencia entre 1Mhz y 25 Mhz (se selecciona usando un cristal externo).
- Oscilador RTC
  - Proporciona un reloj de 1 Hz clock, y una salida de reloj de 32 kHz (que se puede utilizar para la funcion watchdog timer, o para sincronizar la ejecución de instrucciones en la CPU)

2.7 Reloj de CPU

- Estos tres osciladores puede alimentar una etapa de PLL (PLL0) en la que la frecuencia se puede modificar.
  - El oscilador que se conecta al PLL0 se selecciona usando el registro CLKSRCSEL

Table 17. Clock Source Select register (CLKSRCSEL - address 0x400F C10C) bit description

Bit	Symbol	Value	Description	Reset value
1:0	CLKSRC		Selects the clock source for PLL0 as follows:	0
		00	Selects the internal RC oscillator as the PLL0 clock source (default).	
		01	Selects the main oscillator as the PLL0 clock source. Remark: Select the main oscillator as PLL0 clock source if the PLL0 clock output is used for USB or for CAN with baudrates > 100 kBit/s.	
		10	Selects the RTC oscillator as the PLL0 clock source.	
		11	Reserved, do not use this setting.	
		Warning: Improper setting of this value, or an incorrect sequence of changing this value may result in incorrect operation of the device.		
31:2	-	0	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

  - La frecuencia de salida del PLL0 viene definida por
    - Fcco= (2xMxFin)/N
    - M y N se seleccionan por software

2.7 Reloj de CPU

- M y N se selecciona usando el registro el configuracion del PLL0 (PLLOCFG)

Table 20. PLL0 Configuration register (PLL0CFG - address 0x400F C084) bit description

Bit	Symbol	Description	Reset value
14:0	MSEL0	PLL0 Multiplier value. Supplies the value "M" in PLL0 frequency calculations. The value stored here is M - 1. Supported values for M are 6 through 512 and those listed in Table 21. Note: Not all values of M are needed, and therefore some are not supported by hardware. For details on selecting values for MSEL0 see Section 4.5.10 "PLL0 frequency calculation".	0
15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
23:16	NSEL0	PLL0 Pre-Divider value. Supplies the value "N" in PLL0 frequency calculations. The value stored here is N - 1. Supported values for N are 1 through 32. Note: For details on selecting the right value for NSEL0 see Section 4.5.10 "PLL0 frequency calculation".	0
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

- Fcco debe estar en el rango desde 275Mhz a 550 Mhz.
- La frecuencia entregado por PLL0 puede reducirse usando una etapa de division de frecuencia, que se configura mediante el registro CCLKCFG.

2.7 Reloj de CPU

- Registro CCLKCFG

Table 38. CPU Clock Configuration register (CCLKCFG - address 0x400F C104) bit description

Bit	Symbol	Value	Description	Reset value
7:0	CCLKSEL		Selects the divide value for creating the CPU clock (CCLK) from the PLL0 output.	0x00
		0	pllclk is divided by 1 to produce the CPU clock. This setting is not allowed when the PLL0 is connected, because the rate would always be greater than the maximum allowed CPU clock.	
		1	pllclk is divided by 2 to produce the CPU clock. This setting is not allowed when the PLL0 is connected, because the rate would always be greater than the maximum allowed CPU clock.	
		2	pllclk is divided by 3 to produce the CPU clock.	
		3	pllclk is divided by 4 to produce the CPU clock.	
		4	pllclk is divided by 5 to produce the CPU clock.	
		:	:	
		255	pllclk is divided by 256 to produce the CPU clock.	
31:8	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA