ARM (i)

Departamento de Electrónica

Introducción

- Primer ARM desarrollado por Acorn RISC Machine entre 1983-85
  - Creado a partir de un desarrollo de unos estudiantes de Berkeley (Berkeley RISC I)
- En 1990 la empresa se convierte en Advanced RISC Machines Ltd.
  - Compañía inglesa fundada por Apple Computer, Acorn Computer Group y VLSI Technology. Actualmente ARM Limited
- Dedicada al desarrollo de procesadores RISC
- Una de las compañías con gran presencia mundial
- Cubre aprox. el 75 % del mercado mundial en procesadores RISC empotrados

Introducción

- ARM Ltd no hace procesadores, solo los diseña.
- Concede licencias de diseño a diversos fabricantes.
- Por ejemplo, la tecnología ARM está licenciada a varias compañías como: Philips, Atmel, Frescale (fabrica los productos que antes fabricaba Motorola), Cirrus Logic, Hyundai, Intel, Oki, Samsung, Sharp, Lucent, 3Comp, HP, IBM, Sony, …

Introducción

- Microprocesador RISC
- 37 registros de 32 bits (17 o 18 visibles)
- Memoria caché (dependiendo de la aplicación)
- Bus tipo Von Neuman (ARM7)
- Bus tipo Harvard (ARM9 y posteriores)

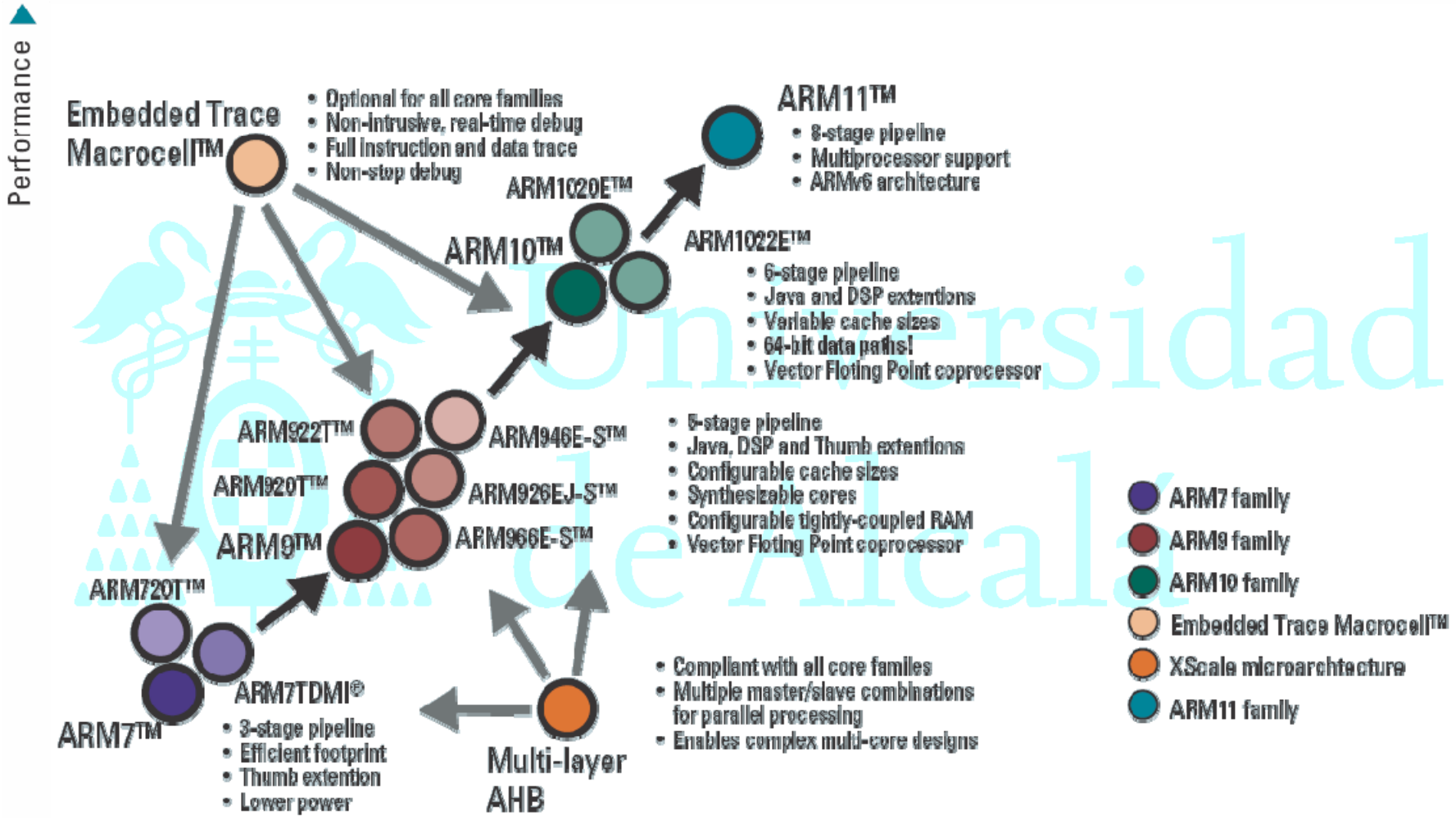
Introducción

- Tipos de datos de 8/16/32 bits
- 6 o 7 modos de operación
- Estructura simple
  - Alta velocidad
  - Bajo consumo de potencia
- Programas compatibles entre las distintas familias ARM

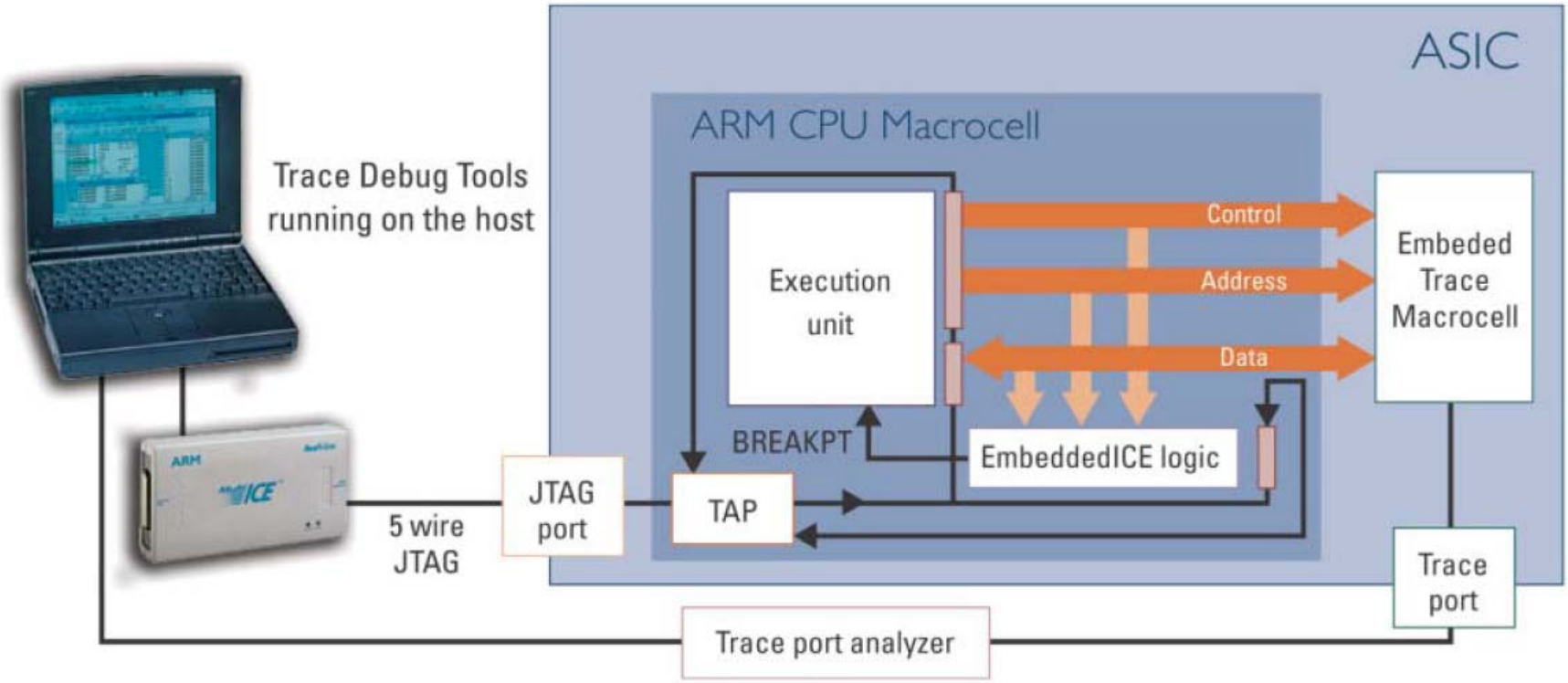
Introducción

- Instrucciones conceptualmente simples
- Transferencias Memoria/Registros, Load/Store
- Operaciones aritméticas entre registros
- Tamaño de instrucciones uniforme (32 bits)
- Pocos formatos para las instrucciones
- Pocos modos de direccionamiento

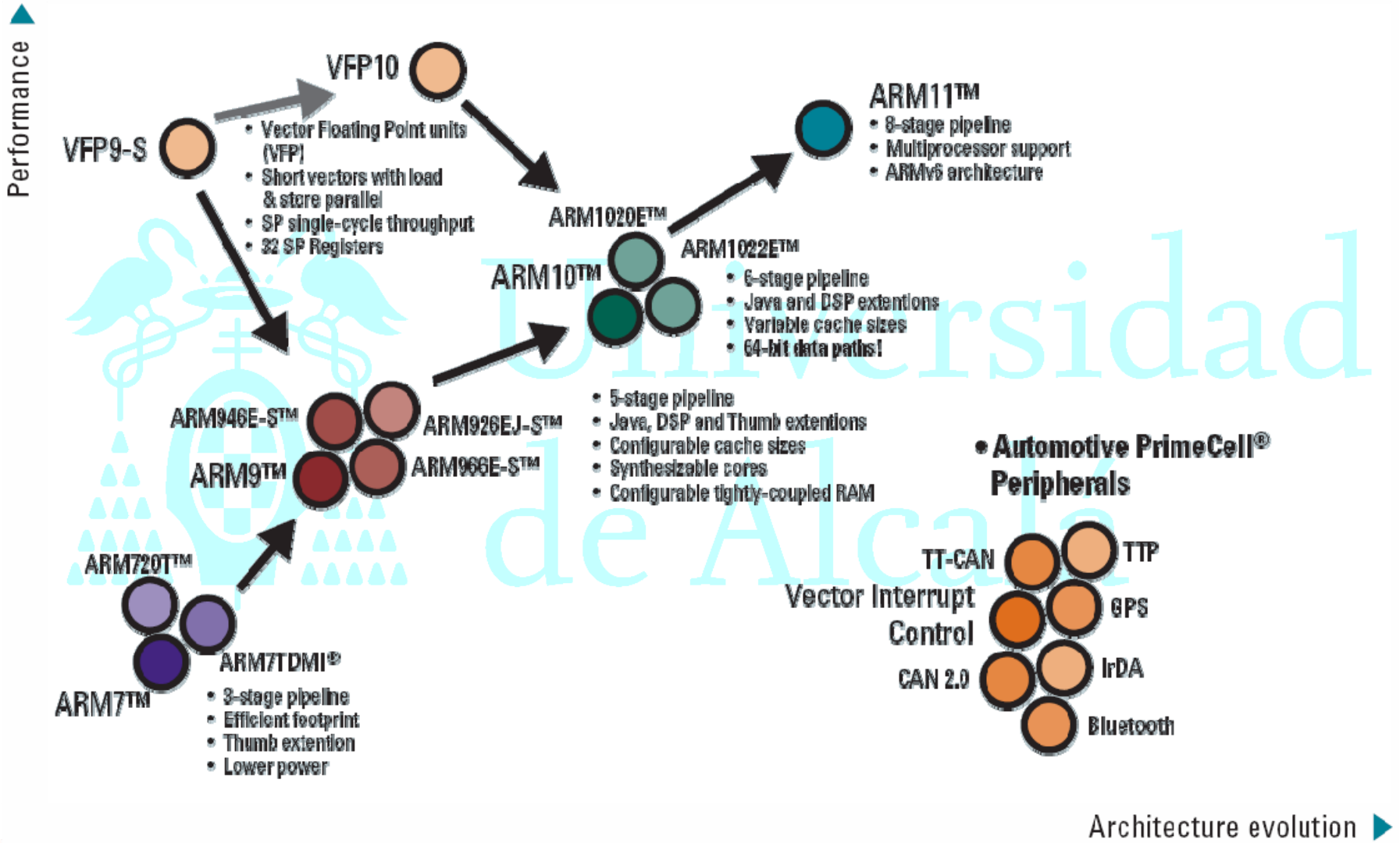
Introducción: Versiones



Introducción: Versiones



Introducción: Versiones (Automoción)



Introducción: Aplicaciones

Introducción: Aplicaciones (Automoción)

ARM technology is well suited for a wide variety Automotive of applications		OSEK Operating System Partners For Automotive
• Fuel injection	• X-by-wire	• ETAS GmbH
• Ignition	• Infotainment	• Euros Plus/Dr. Kaneff
• Transmission	• Navigation	• Integrated Systems
• Power assisted steering	• Body control	• Oki Semiconductors
• Anti-lock brakes		• Realogy
• Airbag, driver information		• Vector Informatik GmbH
• Clutch control		• WindRiver
		• 3soft

Introducción: Aplicaciones (Automoción)

Vendor	Part Number	ARM Core	Application
Micronas	CDC 32xx6	ARM7	Dashboard
ATMEL		ARM7	Fingerprint recognition
ATMEL		7	Airbag
Intel	XScale	StrongARM	Infotainment
Oki	ML67X	7	Powertrain/Body
Philips		426/920[ARM2]	Infotainment
Sharp	LH77790B	7	Telematics
Hynix	GMS30/HMS30	7	Telematics/GPS
ST Microelectronics		7/9	Body/ABS/Airbag/Powertrain/Infotainment
Texas Instruments	TMS470	7	Braking System

Introducción: Periféricos integrados

- Dual Master DMA controller
- DRAM Controller
- Static Memory Controller
- Vectored Interrupt Controller
- UART
- GPIO
- Real-time Clock (RTC)
- Synchronous Serial Port (SSP)
- DC-DC controller
- Advanced Audio Codec Interface (AACI)
- LCD Controller
- Media Card Interface

ARM7: Introducción

- Familia de procesadores ARM
- Versiones del ARM7
  - ARM7TDMI
  - ARM7TDMI-S
  - ARM7EJ-S
  - ARM720T

ARM7: Introducción

- Microprocesadores de propósito general de 32 bits
- Arquitectura RISC
- Número de transistores: 74,209
- Frecuencias de operación: 100 – 133 MHz
- Bus de 32 bits para datos e instrucciones
- Elevado rendimiento: hasta 120 MIPS
- Elevada densidad de código

ARM7: Introducción

- Alimentación: 3.3 V y 5 V
- Bajo consumo de potencia: 80 mW
- Tecnología CMOS
- Extensiones: Thumb, Jazelle
- Los miembros de ARM7 tienen un coprocesador de interfaz que permite una fácil conexión hasta con 16 procesadores más

ARM7: Versiones, ARM7TDMI

- Versión más popular de ARM7
- T: *Thumb*, extensión de instrucciones THUMB
- D: *Debug-interface*
- M: *Multiplier*, multiplicador hardware de alta resolución
- I: *fast Interrupt*

ARM7: Versiones, ARM7TDMI

- 3-stage pipeline
- Efficient footprint
- Thumb extension
- Lower power
- Multi-layer AHB (Advance High-performance Bus)
  - Multiple master/slave combinations for parallel processing
  - Enables complex multi-core designs

ARM7: Versiones, ARM7TDMI

- Embedded Trace Macrocell™
  - Non-intrusive, real-time debug
  - Full instruction and data trace
  - Non-stop debug

ARM7: Versiones, ARM7TDMI-S

- Versión del ARM7TDMI, con los mismos niveles de rendimiento y características en conjunto
- Optimizado para las tendencias modernas de diseño donde portabilidad y flexibilidad son clave
- Recorta el tiempo de entrega al mercado, reduciendo el tiempo de desarrollo a la vez que aumenta la flexibilidad en diseño

ARM7: Versiones, ARM7EJ-S

- Incorpora las ventajas del ARM7TDMI.
- Soporta ejecución acelerada de Java y operaciones DSP.
- Emplea tecnología ARM Jazelle.

ARM7: Versiones, ARM720T

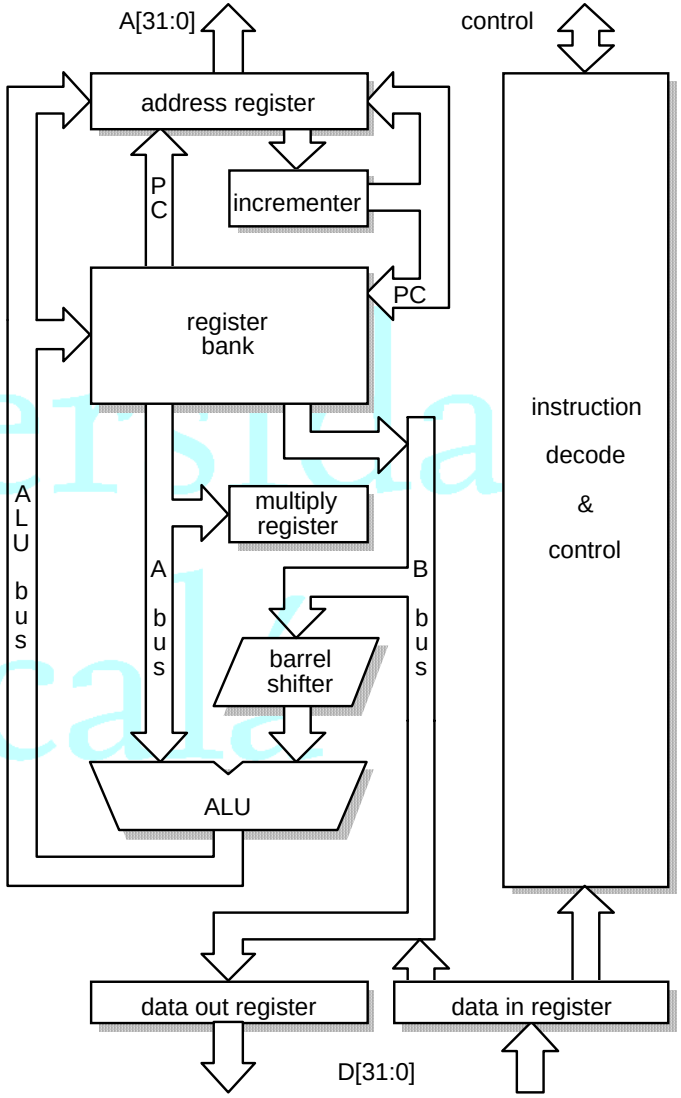
- Para sistemas que requieren manejo completo de memoria virtual y espacios de ejecución protegidos
- Memoria caché de 8K
- MMU: unidad controladora de memoria
- Para aplicaciones de plataforma abierta como Windows CE, Linux, Palm OS y Symbian OS

ARM7: Versiones, ARM720T

- Buffer de escritura
- Bus de interfaz AMBA AHB
- Coprocesador de interfaz ETM para expansión del sistema y depuración en tiempo real
- Coprocesador para control interno de la memoria caché y la MMU
- Acceso rápido a memoria externa

ARM7: Esquema interno

- Register bank
  - Dos puertos de lectura + uno de escritura de acceso a todos los registros
  - 1 puerto de lectura + 1 de escritura adicionales para acceder a R15, PC
- Barrel shifter
  - Desplaza o rota el contenido de los registros
- Address register + incrementer
  - Almacena las direcciones de acceso a memoria
- Data register
  - Almacena los datos en las transferencias con memoria
- Instruction decode & control

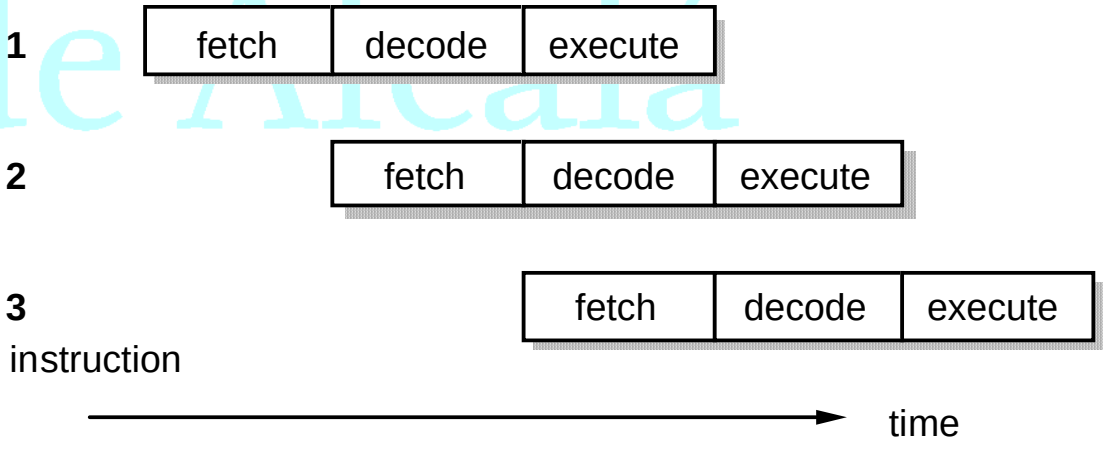


ARM7: Pipeline

- Pipeline de 3 etapas
  - Fetch: Se obtiene la instrucción de memoria y se almacena en el pipeline de instrucciones
  - Decode: Se decodifica la instrucción y se activan las señales de habilitación del "Datapath"
  - Execute: Operaciones y transferencias de datos ocupando el "Datapath"

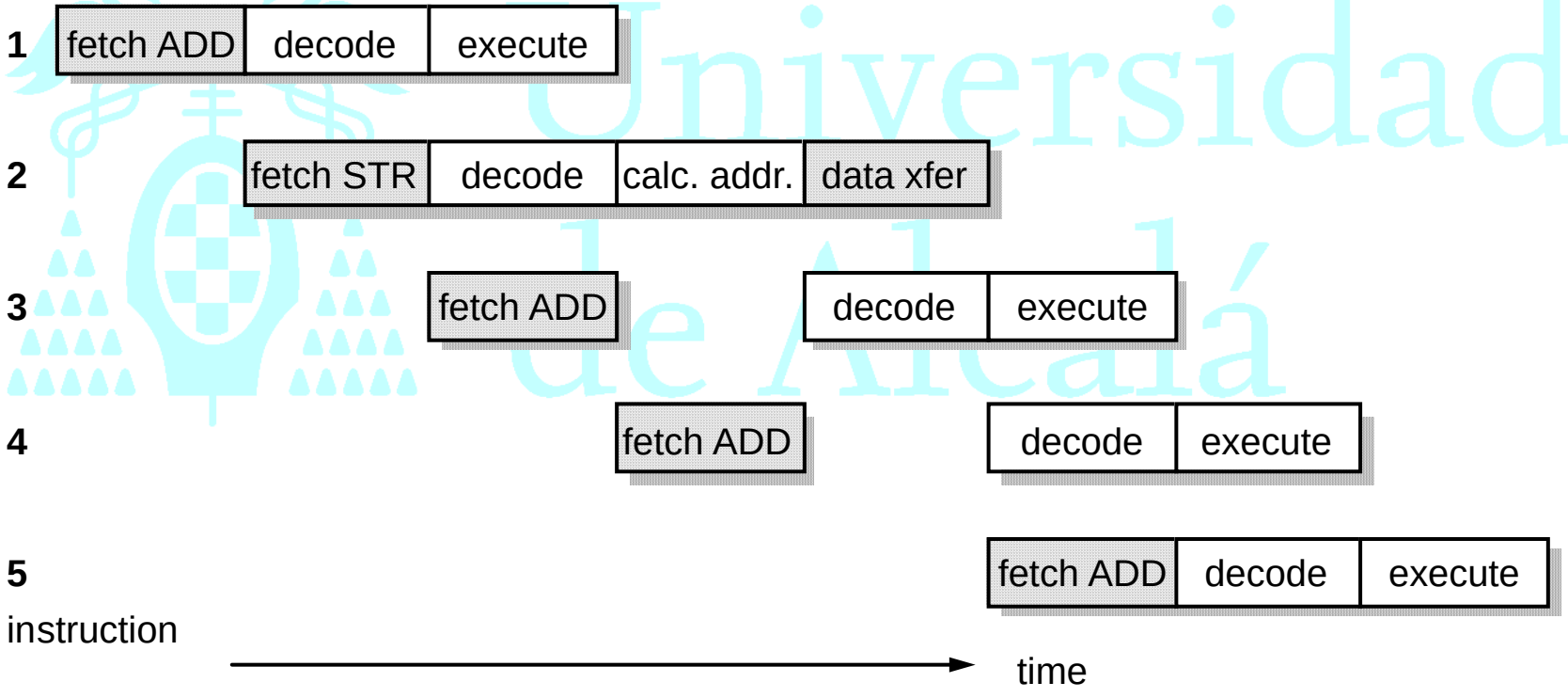
1 instrucción 3 ciclos

1 ciclo 1 instrucción



ARM7: Pipeline

- Pipeline en instrucciones multiciclo
  - Se da cuando una instrucción requiere más de 3 ciclos para ejecutarse. Ejemplo:



ARM7: Pipeline

- El pipeline consta como se ha visto de tres partes
- Todas las instrucciones ocupan el datapath durante al menos un ciclo
- El ciclo anterior activan las líneas de decodificación
- Durante el primer ciclo de datapath se obtiene de memoria la siguiente instrucción

ARM7: Pipeline

- Una instrucción de salto provoca el borrado del pipeline y un nuevo llenado
- En el tercer ciclo de una instrucción se está cargando la segunda instrucción posterior
  - *El PC tiene un valor 8 posiciones superior*

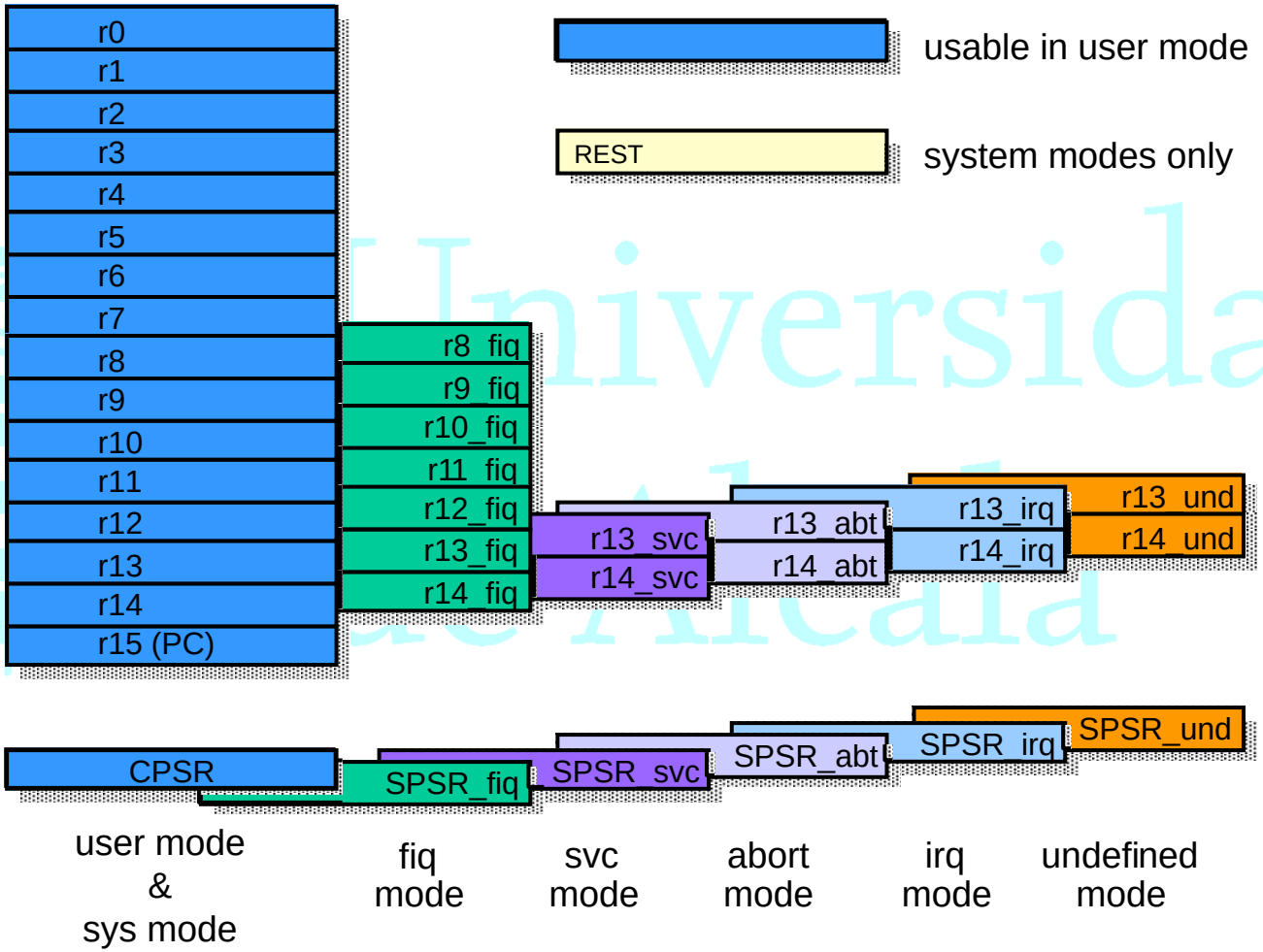
ARM7: Pipeline

- Tiempo de ejecución de un programa
  - $T_{Ejec.} = N_{inst} \cdot CPI_{Med} \cdot T_{CK}$
- Mejora del tiempo de ejecución:
  - Reducir $T_{CK} \Rightarrow$ Simplificar el numero de operaciones por ciclo
  - Reducir CPI $\Rightarrow$ Evitar solapes en los accesos a memoria
- Solución:
  - Arquitecturas Harvard
  - Pipeline de 5 etapas
    - Fetch, Decode, Execute, Buffer/Data, Write-back

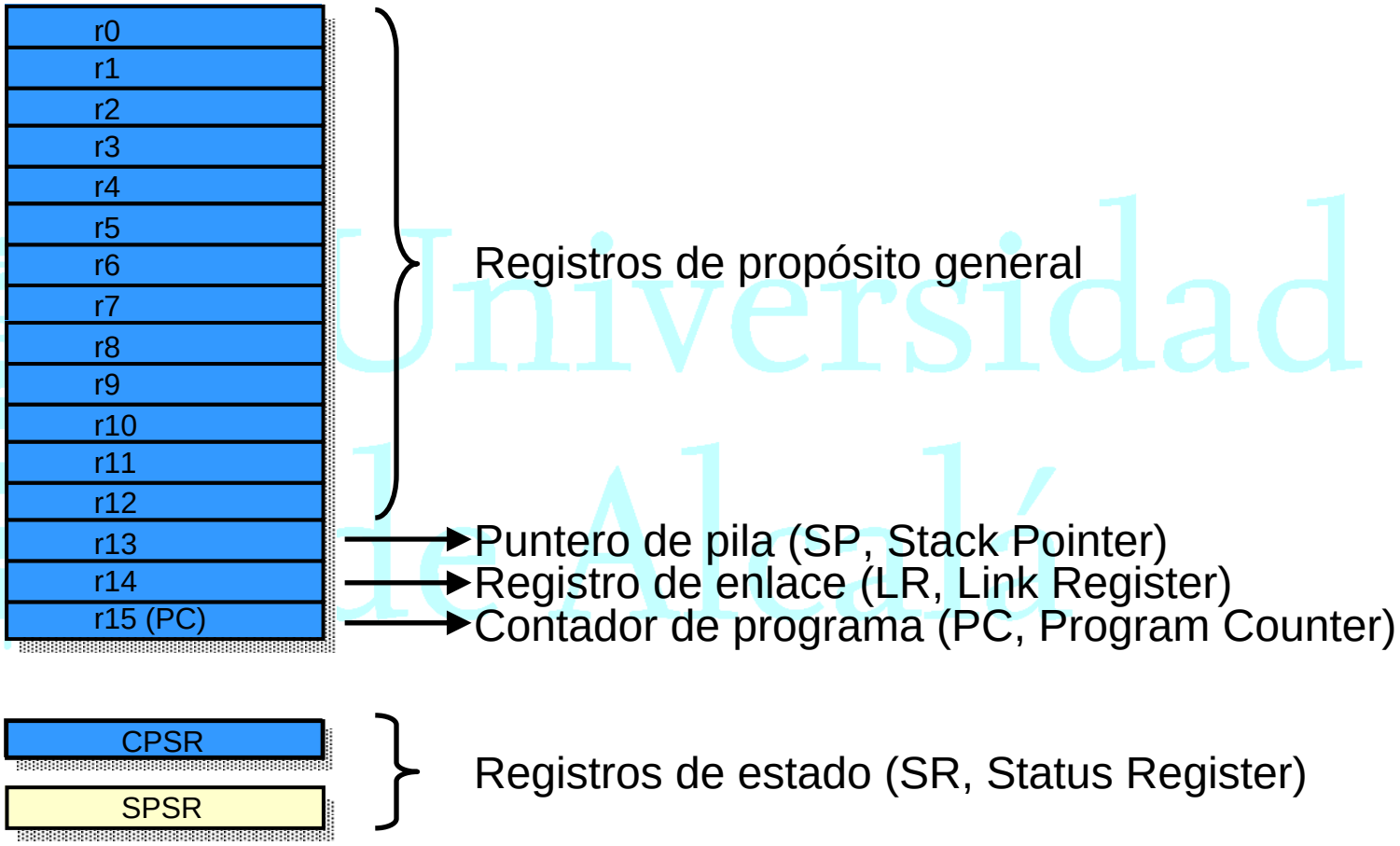
ARM7: Modos de funcionamiento

- User (usr): Estado normal de ejecución de programas
- FIQ (fiq): Empleado en transferencias de alta velocidad
- Supervisor (svc): Modo protegido para el sistema operativo
- Abort mode (abt): Usado cuando se aborta el ciclo fetch de datos o instrucciones
- IRQ (irq): Interrupciones de propósito general
- Undefined (und): Emulación software
- System (sys): Para ejecutar tareas del sistema operativo en modo privilegiado

ARM7: Modelo de programación



ARM7: Registros



ARM7: Registros

- General-purpose Registers
  - Unbanked Registers R0-R7
    - Son comunes a todos los modos de funcionamiento
  - Banked Registers R8-R14
    - Se accede a distintos registros dependiendo del modo en que se encuentre
    - R8-R12, two banked physical registers
    - R13-R14, six banked physical registers

ARM7: Registros

- R13 es el SP (Stack Pointer) por convenio
  - No hay instrucciones específicas, excepto en el modo Thumb
  - Se emplea para al manejo de la pila
  - Cada modo de funcionamiento tiene su propio SP que debe ser inicializado a una zona de memoria dedicada

ARM7: Registros

- R14 es el LR (Link Register)
  - Existe uno para cada modo de funcionamiento
  - Permite almacenar la dirección de retorno en una llamada a una rutina
  - Evita el almacenamiento del contador de programa en la pila
  - Proporciona un retorno rápido
  - Se emplea también para almacenar la dirección de retorno cuando se producen excepciones

ARM7: Registros

- R15, es el PC (Program Counter) contador de programa
  - Contiene la dirección de la instrucción que se va a ejecutar
  - Se puede emplear como cualquier otro registro de propósito general salvo determinadas excepciones
  - La lectura del contador de programa mediante una instrucción devuelve el valor de la dirección actual + 8 bytes
  - Debe tenerse en cuenta para realizar direccionamientos relativos
  - La escritura del PC provoca un salto a la dirección cargada
  - La dirección almacenada debe ser múltiplo de 4, ya que las instrucciones son tamaño word. (Dirección par en modo Thumb)

ARM7: Registros

- Registro de estado CPSR (Current Program Status Register)
  - Accesible en todos los modos de funcionamiento
  - Contiene los flags de condición
- SPSR (Saved Program Status Register)
  - Contiene una copia del CPSR cuando se entra en un modo privilegiado
  - Tienen este registro todos los modos excepto User y System

ARM7: Registros

- Formato del CPSR y del SPSR

- Flags de condiciones
  - N: Bit de signo, 0 ⇒ positivo, 1 ⇒ negativo
  - Z: Bit de cero,
    - 0 ⇒ Resultado de la operación ≠ 0
    - 1 ⇒ Resultado de la operación 0
  - C: Bit de acarreo
  - V: Bit de desbordamiento
  - Q: Desbordamiento/Saturación (bit 27 en versiones E)

ARM7: Registros

- Bits de control

  - Se modifican durante la ejecución de una excepción o por programa
- Bits de habilitación de interrupciones
  - I, puesto a 1 deshabilita las interrupciones IRQ
  - F , puesto a 1 deshabilita las interrupciones FIQ
- Bit T, selección del conjunto de instrucciones Thumb
  - T, 0 ejecución ARM, 1 ejecución Thumb
  - En versiones sin extensión Thumb funciona como bit de traza

ARM7: Registros

- Bits de modo

  - M0, M1, M2, M3 y M4 representan el modo de operación del micro

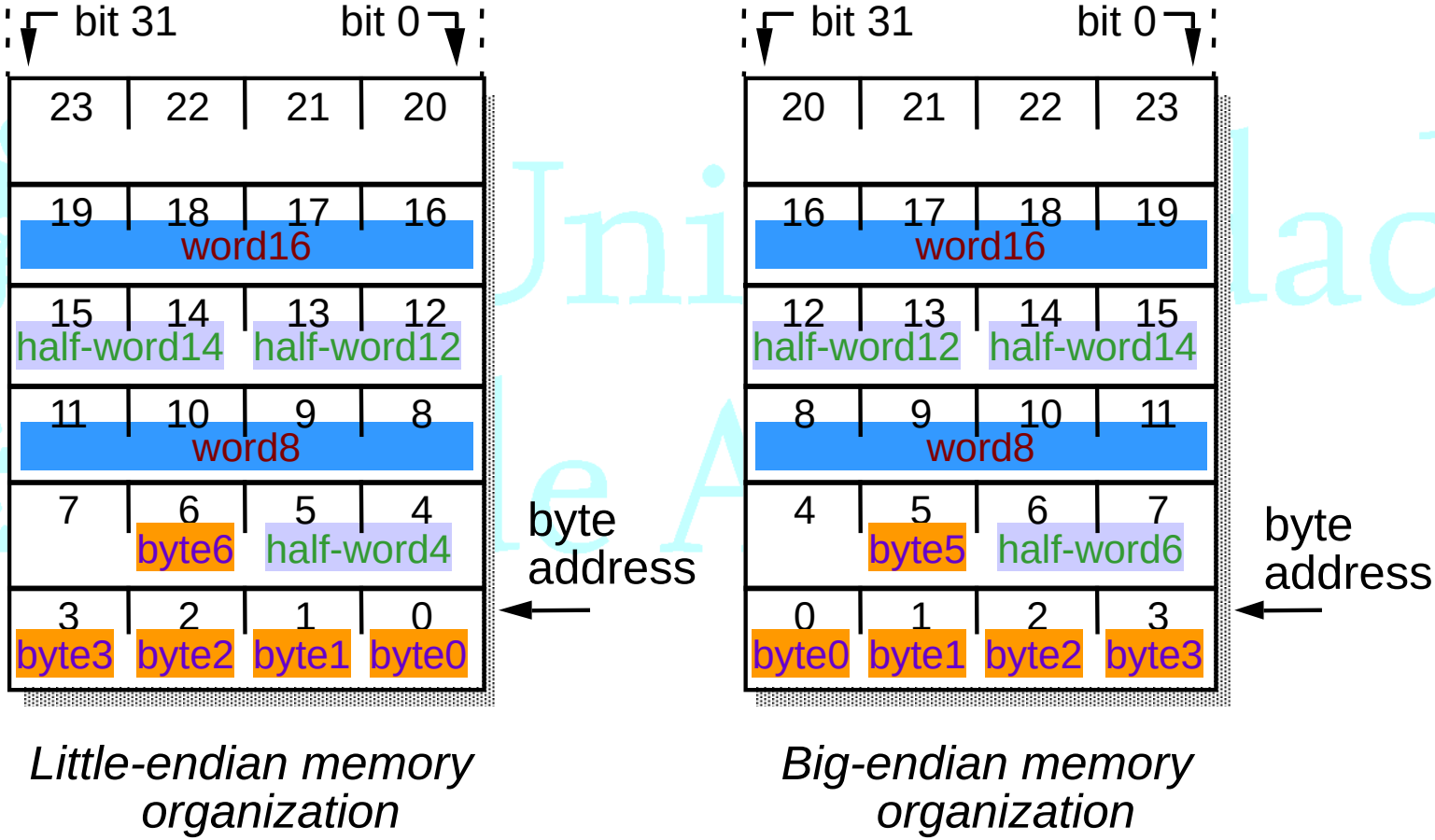
CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

ARM7: Tipos de datos

- Byte, 8 bits
- Halfword, 16 bits
- Word, 32 bits
- Representaciones
  - Sin signo $\Rightarrow$ 0 .. 2^N-1, Con signo $\Rightarrow$ C2, -2^{N-1} .. 2^{N-1}-1
- Almacenamiento en registros y operaciones en tamaño word
  - Posibilidad de extensión de signo
- Transferencias con memoria en todos los tamaños

ARM7: Datos en memoria

- ARM memory organization

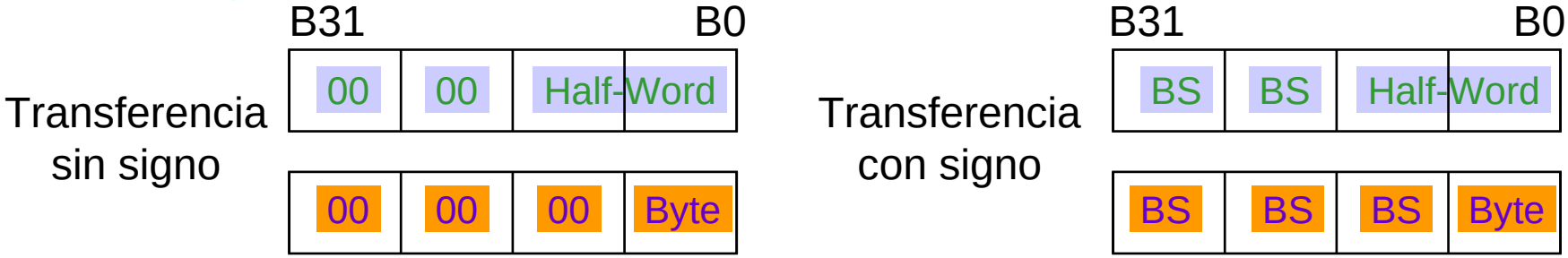


Little-endian memory organization

Big-endian memory organization

ARM7: Datos en memoria

- Organización de los datos en los registros
  - Las transferencias de datos de memoria a registros pueden ser en tamaño byte, word y halfword
  - La transferencia de datos en tamaño inferior a word se almacenan en la parte de menor peso de los registros, rellenado con 0 la parte alta
  - Existen instrucciones para transferencia de bytes o halwords con signo, en cuyo caso los bits de mayor peso serán una extensión del signo de dato transferido



ARM7: Instrucciones

- Se puede hacer una división simple en:
  - Instrucciones de proceso de datos
  - Instrucciones de transferencia de datos
  - Instrucciones de control

ARM7: Instrucciones de Proceso de Datos

- Instrucciones de Proceso de Datos
  - Permiten realizar operaciones aritméticas y lógicas sobre el contenido de los registros
  - Típicamente requieren dos operandos y generan un único resultado sencillo
    - Por ejemplo sumar dos valores. (Existen excepciones a ambos casos)
- Características de las instrucciones de proceso de datos
  - Operandos fuente y destino de 32 bits. (Excepto "long multiply", 64 bits)
  - Los operandos son siempre registros o valores inmediatos
  - Se pueden especificar los operandos fuente y destino de forma independiente
  - Ejemplo:
    - Suma r1+ r2 y almacena el resultado en r0
      - ADD r0, r1, r2 ; r0 operando destino, r1 y r2 operandos fuente

ARM7: Instrucciones de Proceso de Datos

- Instrucciones Aritméticas
  - Suma binaria
  - Resta binaria
  - Resta binaria inversa

```
ADD   r0, r1, r2      ; r0 = r1 + r2
ADC   r0, r1, r2      ; r0 = r1 + r2 + C  ⇒ Suma con acarreo
SUB   r0, r1, r2      ; r0 = r1 - r2
SBC   r0, r1, r2      ; r0 = r1 - r2 + C -1 ⇒ Resta con acarreo
RSB   r0, r1, r2      ; r0 = r2 – r1  ⇒ Resta binaria inversa
RSC   r0, r1, r2      ; r0 = r2 – r1+ C -1 ⇒ Resta binaria inversa con acarreo
```

ARM7: Instrucciones de Proceso de Datos

- Instrucciones lógicas
  - Producto lógico
  - Suma lógica
  - Suma exclusiva
  - Borrado de un bit

```
AND   r0, r1, r2      ; r0 = r1 and r2,      r0 = r1 & r2;
ORR   r0, r1, r2      ; r0 = r1 or r2,       r0 = r1 | r2;
EOR   r0, r1, r2      ; r0 = r1 xor r2,      r0 = r1 ^ r2;
BIC   r0, r1, r2      ; r0 = r1 and not r2,  r0 = r1 & !r2  ; // Blt Clear
```

ARM7: Instrucciones de Proceso de Datos

- Instrucciones de movimiento de datos
  - Se diferencian de las de transferencia en que estas se realizan entre registros, y las otras con memoria. Se consideran de proceso de datos porque los modifican
  - Omiten el primero operando fuente
  - Copian el segundo operando de forma directa o en complemento a 1

```
MOV     r0, r2          ; r0 = r2
MVN     r0, r2          ; r0 = not r2,          r0 = ! r2; // MoVe Negated
```

ARM7: Instrucciones de Proceso de Datos

- Instrucciones de comparación
  - No generan resultado, sólo posicionan los códigos de condición (cc) del registro de estado

```
CMP     r1, r2      ; Posiciona los cc en función del resultado de la operación (r1 - r2)
CMN     r1, r2      ; Posiciona los cc en función del resultado de la operación (r1 + r2)
TST     r1, r2      ; Posiciona los cc en función del resultado de la operación (r1 & r2)
TEQ     r1, r2      ; Posiciona los cc en función del resultado de la operación (r1 ^ r2)
```

  - CMP (CoMPare), CMN (CoMpare Negated), TST (TeST), TEQ (Test EQual)

ARM7: Instrucciones de Proceso de Datos

Operandos inmediatos

- Las instrucciones anteriores, además de con registros pueden operar con valores inmediatos

```
ADD     r0, r1, #5          ; r0 = r1 + 5; // El valor inmediato va precedido de #
ADD     r3, r3, #1          ; r3 ++;
AND     r8, r7, #&FF        ; r8 = r7 & 0xFF; // Ojo en ensamblador del ARM "&"
                            ; significa hexadecimal, equivale a "0x" en C
```

- Los valores inmediatos se codifican en la misma instrucción, por lo que están limitados en tamaño
  - Valor inmediato = $(0 .. 255) \cdot 2^{2n}$, $0 \le n \le 12$
- Cubre todas las potencias de 2, pero hay valores que no se pueden codificar

ARM7: Instrucciones de Proceso de Datos

Instrucciones de desplazamiento y rotación

- No existen como tales, pero se pueden emplear para modificar un operando antes de realizar una de las operaciones anteriores

```
ADD     r1, r2, r3, LSL #5      ; r1 = r1 + r2 + r3 * 32 ; r3 se desplaza 5 veces a la
                                ; izquierda, lo que equivale a añadirle 5 ceros a la derecha,
                                ; que es equivalente a multiplicar por 2^5= 32
```

- A pesar de lo complejo de la instrucción el tiempo de ejecución es el mismo que las instrucciones simples (1 ciclo de reloj)
- Es posible indicar el numero de desplazamientos mediante un registro

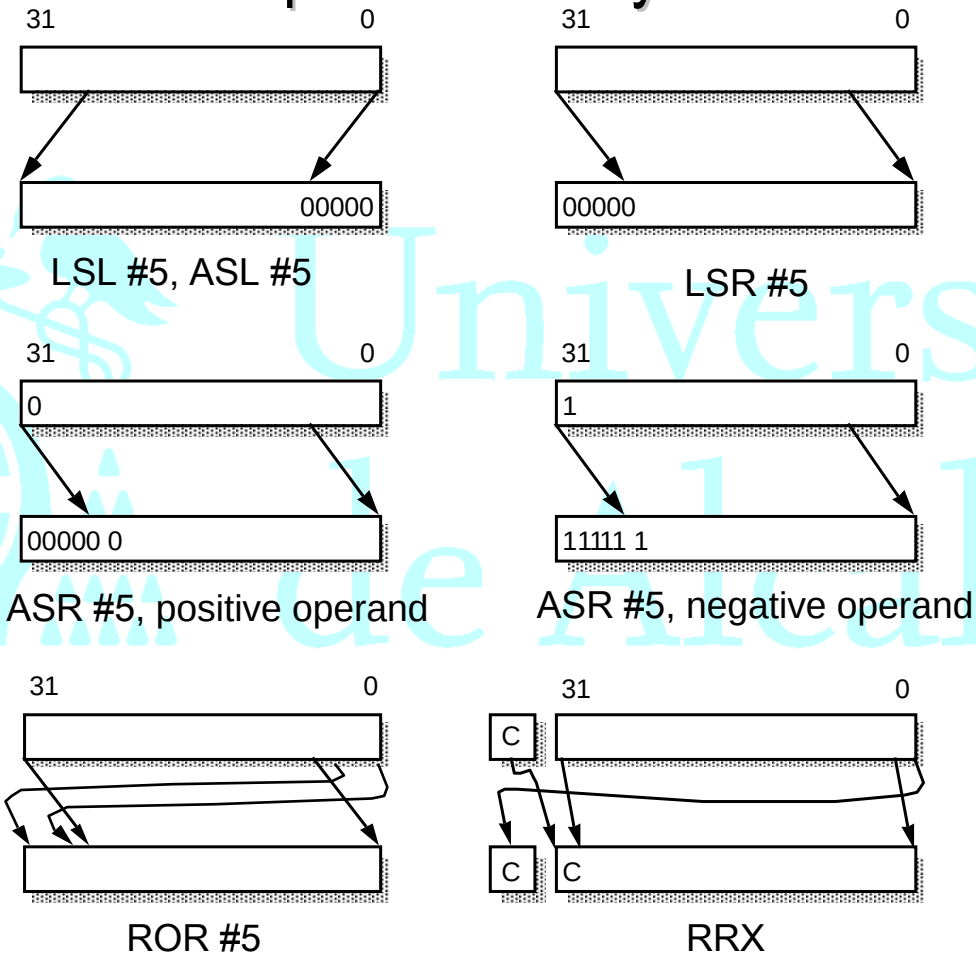
```
ADD     r1, r2, r3, LSL r4      ; r1 = r1 + r2 + r3 * 2^r4
```

ARM7: Instrucciones de Proceso de Datos

- Operaciones de desplazamiento y rotación disponibles
  - LSL, Logical Shift Left, desplazamiento lógico a la izquierda, añade ceros por la derecha
  - LSR, Logical Shift Right, desplazamiento lógico a la derecha, añade ceros por la izquierda
  - ASL, Arithmetic Shift Left, desplazamiento aritmético a la izquierda, añade ceros por la derecha, es igual que LSL
  - ASR, Arithmetic Shift Right, desplazamiento aritmético a la derecha, repite el signo en los bits de la izquierda
  - ROR, ROtate Rigth, rotación a derechas. El bit que se pierde se realimenta
  - RRX, Rotate Rigth eXtended, rotación a derechas extendida al flag de acarreo

ARM7: Instrucciones de Proceso de Datos

- Operaciones de desplazamiento y rotación disponibles



ARM7: Instrucciones de Proceso de Datos

Códigos de condición

- Las instrucciones de comparación posicionan siempre los códigos de condición
- El resto de instrucciones pueden posicionarlos o no a elección del programador
- Esto se consigue añadiendo el sufijo “s” al nemónico correspondiente

```
ADDS    r2, r2, r0        ; Suma y posiciona los flags de cc
```

- Las operaciones de comparación y aritméticas posicionan todos los flags
- Las operaciones lógicas sólo posicionan los flags N y Z, el resto permanecen intactos, excepto que se realice una rotación extendida, en cuyo caso modifican el flag C

ARM7: Instrucciones de Proceso de Datos

- Instrucciones de multiplicación
  - No soporta como segundo operando un inmediato
  - El registro de resultado no puede coincidir con el primer operando

```
MUL     r1, r2, r3        ; r1= r2 * r3
```

  - El resultado de la operación es en 64, pero el almacenamiento se realiza en un registro de 32. El valor almacenado consiste en los 32 bits bajos del resultado
  - Existe una variante de multiplicación y acumulación

```
MLA     r1, r2, r3, r4    ; r1= r2 * r3 + r4
```

ARM7: Instrucciones de Proceso de Datos

- Si se activa la opción “s” el flag V no se afecta, y el C se modifica pero no tiene significado
- Para multiplicar por una constante se puede cargar la constante en un registro y a continuación multiplicar por el registro. Ejemplo: multiplicar r0*27 y almacenar el resultado en r0

```
MOV     r1, #27
MOV     r2, r0
MUL     r0, r1, r2      ; r0= r0 * 27
```

- En estos casos es más eficiente el empleo de instrucciones de suma con desplazamiento

```
ADD     r0, r0, r0, LSL #1      ; r0= r0 * 3
ADD     r0, r0, r0, LSL #3      ; r0= r0 * 9
```

ARM7: Instrucciones de Transferencia

- Instrucciones de transferencia
  - Mueven datos entre registros y memoria
  - Load (memoria a registro) / Store (registro a memoria)
- Tipos básicos
  - Transferencia desde o hacia un único registro
    - Admite tamaños de operandos Byte (8 bits), Word (32 bits) o Halfword (16 bits)
  - Transferencia desde o hacia múltiples registros
    - Se emplean para salvar o recuperar el contenido de varios registros en memoria
  - Instrucciones de intercambio, SWAP
    - Permite el intercambio de los datos entre un registro y una posición de memoria. Realiza una operación simultánea de load y store

ARM7: Instrucciones de Transferencia

- Modos de direccionamiento
  - Las formas de acceder a memoria son variadas, y están definidas por los diferentes modos de direccionamiento
  - La transferencia de los datos de memoria implican un registro, por lo que uno de los modos de direccionamiento empleados será directo a registro
  - Los accesos a memoria están basados todos ellos en direccionamiento indirecto por registro, en sus distintas variantes

ARM7: Instrucciones de Transferencia

- Direccionamiento indirecto por registro
  - La dirección efectiva es la contenida en el registro empleado

```
LDR     r0, [r1]        ; Carga en r0 el contenido de la posición apuntada por r1
STR     r0, [r1]        ; Almacena el contenido de r0 en la posición apuntada por r1
```

- Inicialización del registro índice
  - Para asignar el valor inicial del registro índice se puede emplear una instrucción de movimiento de datos
  - Otra alternativa es emplear una pseudo instrucción del ARM, que carga un valor en el registro empleando como base el valor del PC

```
ADR     r1, Tabla1      ; Hace que r1 apunte a Tabla1. ADR se traduce por una instrucción
                        ; de suma o resta
```

ARM7: Instrucciones de Transferencia

- Ejemplo de transferencia sencilla

```
ADR     r1, Tabla1      ; r1 apunta a Tabla1
ADR     r2, Tabla2      ; r2 apunta a Tabla2
LDR     r0, [r1]        ; Carga en r0 el contenido de la posición Tabla1
STR     r0, [r2]        ; Almacena el contenido de r0 en la posición Tabla2
```

- Para recorrer la tabla se pueden incrementar los registros mediante una instrucción de suma

```
ADD     r1, r1, #4      ; r1 apunta al siguiente word de la Tabla1, cada word ocupa 4
                        ; posiciones
ADD     r2, r2, #4      ; r2 apunta al siguiente word de la Tabla2
```

ARM7: Instrucciones de Transferencia

- Direccionamiento indirecto por registro con desplazamiento
  - La dirección efectiva es la contenida en el registro empleado sumado con el valor inmediato adjunto

    ```
    LDR     r0, [r1, #4]     ; Carga en r0 el contenido de la posición apuntada por r1+4
    ```

  - En el ARM este modo es conocido como “direccionamiento preindexado”
  - Al igual que en el ejemplo anterior, puede ser necesario modificar el valor de r1, esto puede realizarse empleando un modo de “direccionamiento preindexado con autoindexado”

    ```
    LDR     r0, [r1, #4]!    ; Carga en r0 el contenido de la posición apuntada por r1+4 e
                             ; incrementa el contenido de r1 en 4
    ```

  - El incremento de r1 no requiere un tiempo de ejecución adicional

ARM7: Instrucciones de Transferencia

- Direccionamiento postindexado
  - Realiza el incremento del registro, pero no emplea el desplazamiento en el direccionamiento

```
LDR     r0, [r1], #4    ; Carga en r0 el contenido de la posición apuntada por r1 e
                        ; incrementa el contenido de r1 en 4
```

- Selección del tamaño y modo de transferencia
  - Como se comentó las transferencias con memoria pueden realizarse en tres tamaños diferentes y con o sin signo. Esto se consigue añadiendo modificadores a las instrucciones ya vistas

```
LDRB    r0, [r1]    ; Transferencia en tamaño Byte sin signo
LDRH    r0, [r1]    ; Transferencia en tamaño Half-Word sin signo
LDRSB   r0, [r1]    ; Transferencia en tamaño Byte con signo
LDRSH   r0, [r1]    ; Transferencia en tamaño Half-Word con signo
```

ARM7: Instrucciones de Transferencia

Transferencia hacia múltiples registros

- Carga el contenido de posiciones consecutivas de memoria en varios registros

```
LDMIA    r0, {r1, r2,r5}    ; Carga en r1 el contenido de la posición apuntada por r0,
                            ; en r2 el apuntado por r0+4 y en r5 el apuntado por r0+8
```

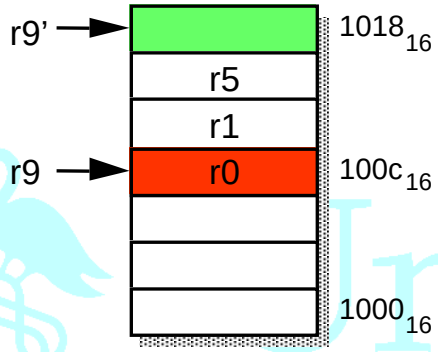
- El orden en que se especifiquen los registros es irrelevante, se carga el dato de la posición mas baja de memoria en el registro de menor orden y el resto en los registros de orden inmediato superior
- Las transferencias se realizan en 32 bits, por lo que r0 debe apuntar a una posición múltiplo de 4

Cuadro resumen de instrucciones de transferencias múltiples

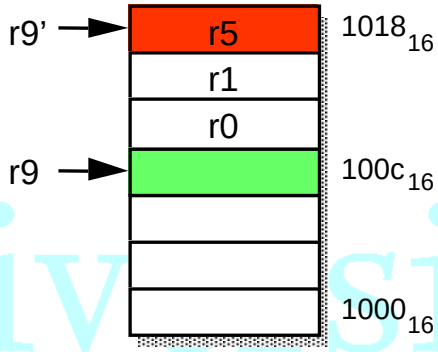
		Ascending		Descending	
		Full	**Empty**	**Full**	**Empty**
Increment	**Before**	STMIB, STMFA			LDMIB, LDMED
	After		STMIA, STMEA	LDMIA, LDMFD	
Decrement	**Before**		LDMDB, LDMEA	STMDB, STMFD	
	After	LDMDA, LDMFA			STMDA, STMED

ARM7: Instrucciones de Transferencia

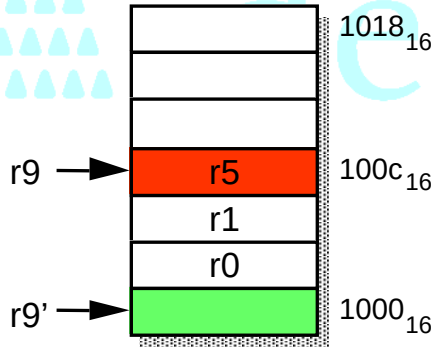
- Instrucciones de transferencias múltiples y su efecto en memoria



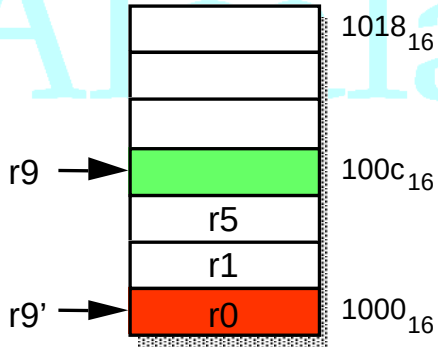
`STMIA r9!, {r0,r1,r5}`



`STMIB r9!, {r0,r1,r5}`



`STMDA r9!, {r0,r1,r5}`



`STMDB r9!, {r0,r1,r5}`

ARM7: Instrucciones de Transferencia

Control de la Pila

- No existen instrucciones específicas para el control de la pila, por lo que se emplean las de transferencia de datos
- Una pila tiene un funcionamiento LIFO (Last Input First Output) por lo que hay que almacenar y recuperar datos en orden inverso
- Las pilas funcionan de diferente modo dependiendo del microprocesador, dado que aquí el control es total, se puede generar una pila que almacene los datos en sentido ascendente o descendente, es decir, que incremente la posición de memoria cada vez que almacene datos o la decremente
- Del mismo modo, el puntero de pila puede apuntar al último dato almacenado o al primer espacio libre dentro de la pila ("full stack" y "empty stack" respectivamente

ARM7: Instrucciones de Transferencia

- Ejemplo de acceso a la Pila
  - Por convenio se emplea el registro r13 como Stack Pointer
  - Almacenamiento de un dato en la pila

```
STR     r0, [r13, #4]!   ; Almacena el contenido de r0 en la dirección r13+4 e incrementa r13
                         ; en 4
```

  - Recuperación de un dato de la pila

```
LDR     r0, [r13], # -4  ; Carga en r0 el contenido de la dirección r13 y decrementa r13 en 4
```

  - En este ejemplo el puntero contiene la dirección del último dato escrito (full stack) y crece hacia posiciones altas de memoria (ascending stack), se trata por tanto de una pila "full ascending"
  - Combinando las instrucciones se pueden conseguir los otros tres modos posibles: "full descending", "empty ascending", "empty descending"

ARM7: Instrucciones de Transferencia

- Ejemplo de acceso a la Pila
  - También se puede realizar el acceso con instrucciones de acceso múltiple

```
STMFD    r0!, {r2-r9}        ; Almacena los registros r2 a r9 en la pila (Full Descending)
....
....
LDMFD    r0!, {r2-r9}        ; Recupera los datos de la pila
```

  - Téngase en cuenta que los modos de almacenamiento y recuperación deben ser los duales para que la pila quede en el estado inicial después de recuperar los datos
  - Esta instrucción, evidentemente, no se ejecuta en un ciclo de reloj ni aun en el caso de arquitecturas Harvard

ARM7: Instrucciones de Control

- Este conjunto de instrucciones permiten determinar que instrucción se ejecutará, pero no realizan modificación ni transferencia de datos
- Dentro de este grupo de instrucciones están las instrucciones de salto
- Tipos de salto
  - Salto incondicional: Se realiza siempre que aparece la instrucción

```
B       Etiqueta        ; Salta a la instrucción señalada por "Etiqueta"
```

  - Salto condicional: Se ejecuta si se cumple una condición basada en los flags del registro de estado

```
BEQ     Etiqueta        ; Salta a la instrucción señalada por "Etiqueta"
                        ; si está activo el flag de cero
```

ARM7: Instrucciones de Control

- Condiciones de salto

Branch	Interpretation	Normal uses
B BAL	Unconditional Always	Always take this branch Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC BLO	Carry clear Lower	Arithmetic operation did not give carry-out Unsigned comparison gave lower
BCS BHS	Carry set Higher or same	Arithmetic operation gave carry-out Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

ARM7: Instrucciones de Control

Ejemplo de empleo

- Ejecución de un conjunto de instrucciones varias veces, equivalente a un bucle “for (i=0; i<10; i++) { ….}”

```
        MOV     r0, #0          ; Inicialización del contador, i=0;
Bucle   ....                    ; { Conjunto de instrucciones
        ....                    ; que se repiten en el bucle }
        ADD     r0, r0, #1      ; Incremento del contador, i++;
        CMP     r0, #10         ; Comparación del contador, i<10;
        BNE     Bucle           ; Salto al principio del bucle mientras se cumpla la condición
```

ARM7: Instrucciones de Control

- Ejemplo de empleo
  - Ejecución condicional de una instrucción “if (r0==5) {r1=2*r1-r2;}”

```
            CMP     r0, #5
            BNE     Siguiente       ; if
            ADD     r1, r1, r1      ; { Instrucciones que se ejecutan
            SUB     r1, r1, r2      ; si se cumple la condición}
Siguiente   ....
```

  - Otra variante

```
            CMP     r0, #5
            BNE     Siguiente       ; if
            RSB     r1, r2, r1, LSL #1  ; { Instrucciones }
Siguiente   ....
```

ARM7: Instrucciones de Control

- Como se comentó anteriormente las instrucciones de salto vacían el pipeline, y ralentizan la ejecución del programa
- Existe otra posibilidad de ejecución condicional
  - Ejecución condicional de una instrucción "if (r0==5) {r1=2*r1-r2;}"

```
CMP     r0, #5
ADDNE   r1, r1, r1      ; { Instrucciones que se ejecutan
SUBNE   r1, r1, r2      ; si se cumple la condición}
....
```

ARM7: Instrucciones de Control

- La estructura del ARM permite ejecutar cada instrucción de forma condicional, reflejando la condición dentro del código de operación

- Ejemplo de ejecución condicional con varias condiciones
  - If ((a==b) && (c==d)) e++;

```
CMP     r0, r1          ; if (a==b)
CMPEQ   r2, r3          ; { if (c==d)
ADDEQ   r4, r4, #1      ; e++;}
....
```

ARM7: Instrucciones de Control

- Llamadas a subrutinas
  - Cuando se realiza la llamada a una rutina hay que almacenar la dirección de retorno, que coincide con la dirección de la instrucción siguiente a la de llamada
  - Muchos microprocesadores almacenan la dirección de retorno en la pila
  - El ARM no tiene control automático de pila, por lo que no almacenará la instrucción de retorno en ella
  - La llamada a una rutina se puede realizar con la instrucción de salto B

ARM7: Instrucciones de Control

- Otro modo de llamar a una subrutina es emplear BL
  - BL, Branch and Link, hace una copia de la dirección de la siguiente instrucción en el registro de enlace, r14
- El ARM no tiene instrucciones de retorno de subrutinas, el retorno se realiza copiando el contenido del r14 en el PC
- Ejemplo de llamada a subrutina

```
        BL      Subrutina  ; Salta a la subrutina y copia la dirección de retorno en LR (r14)
        …                  ; Dirección de retorno
        …
Subrutina …
        …
        …
        MOV     pc, r14    ; Copia el contenido del "Link Register" en el "PC"
```

ARM7: Instrucciones de Control

- Anidamiento de subrutinas
  - Dado que sólo hay un registro r14 en cada modo, si se requiere anidar rutinas, es necesario almacenar su contenido en la pila para recuperarlo posteriormente
  - Ejemplo:

```
            BL      Subrutina_1     ; Llamada a la subrutina
            …
Subrutina_1 STMFD   r13!, {r0-r2, r14}  ; Almacenamiento en pila de los registros que se usan en
                                        ; la Subrutina_1 y de el LR
            …
            BL      Subrutina_2     ; Llamada a la Subrutina_2
            …
            LDMFD   r13!, {r0-r2, r14}  ; Recuperación de los registros
            MOV     pc, r14         ; Retorno de la Subrutina_1
Subrutina_2 …
            …
            MOV     pc, r14         ; Instrucción de retorno de la Subrutina_2
```

ARM7: Instrucciones de Control

- Así mismo, otra opción de retorno es recuperar el contenido del pc directamente desde la pila
  - Ejemplo:

```
Subrutina_1  STMFD     r13!, {r0-r2, r14}  ; Almacenamiento en pila de los registros que se usan en
                                           ; la Subrutina_1 y de el LR
             …
             …
             BL        Subrutina_2
             …
             …
             LDMFD     r13!, {r0-r2, pc}   ; Recuperación de los registros r0, r1 y r2, y retorno de la
                                           ; subrutina
```

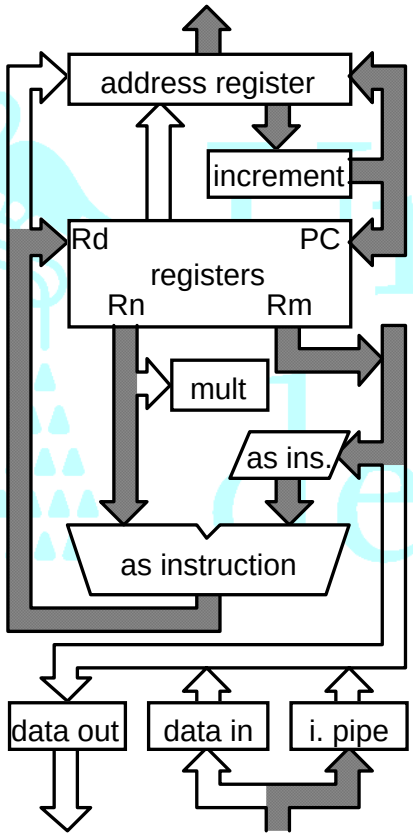
  - Nótese que en la recuperación de los registros se ha sustituido r14 por r15 (PC), por lo que se provoca el retorno de forma automática desde la pila, omitiendo el paso de recuperar el contenido de r14

ARM7: Instrucciones de Control

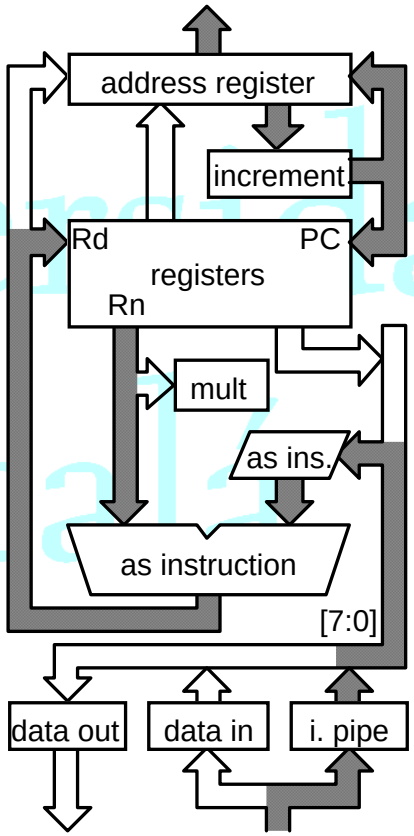
- Ejecución de rutinas en modo supervisor
  - Para ejecutar una rutina en modo supervisor se emplean las interrupciones software "swi", también conocidas como llamadas de supervisor
  - Generalmente realizan llamadas a rutinas del sistema operativo

ARM7: Actividad del Datapath

- Esquemas de ocupación del datapath para los distintos tipos de instrucciones. Instrucciones de procesamiento



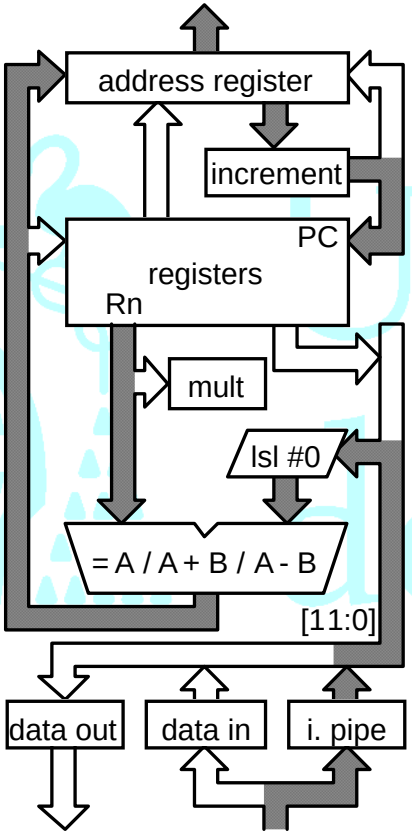
(a) register - register operations



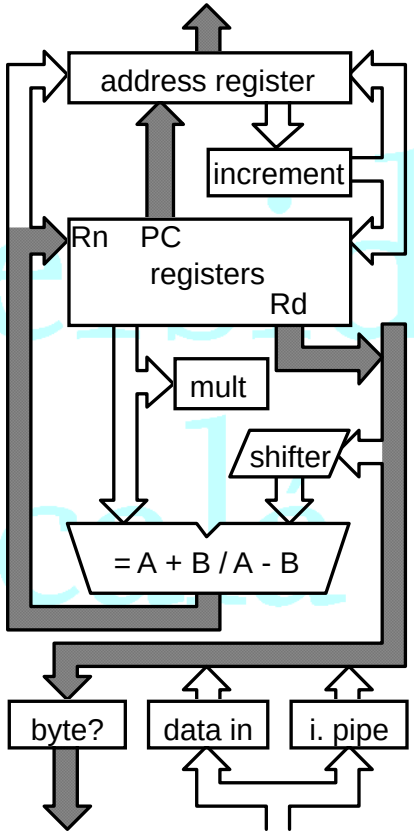
(b) register - immediate operations

ARM7: Actividad del Datapath

- Instrucciones de transferencia (STR)



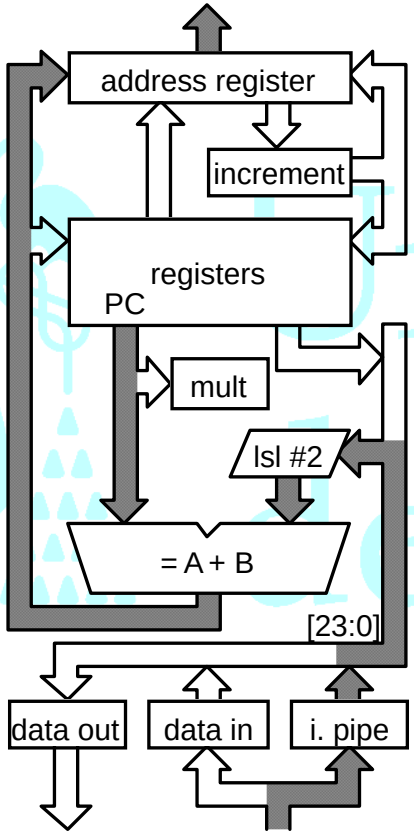
(a) 1st cycle - compute address



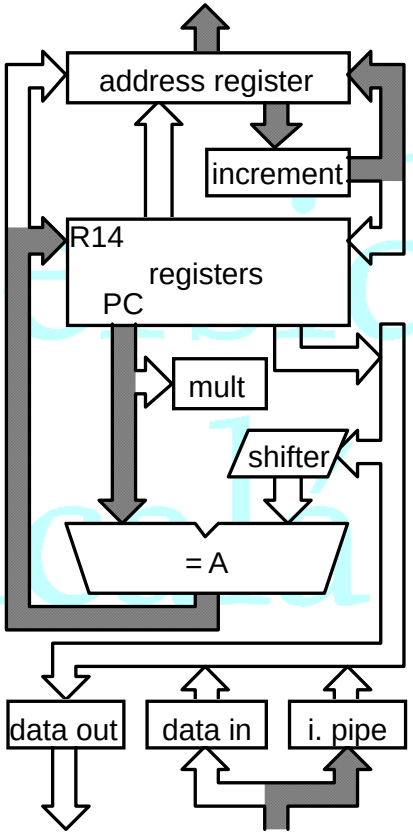
(b) 2nd cycle - store data & auto-index

ARM7: Actividad del Datapath

- Instruccion de salto



(a) 1st cycle - compute branch target



(b) 2nd cycle - save return address

ARM7: Tratamiento de Excepciones

- Las excepciones se emplean para tratar eventos que se producen durante la ejecución de un programa
- En algunos microprocesadores, incluido el ARM, el concepto de excepción incluye las interrupciones hardware y software
- Las interrupciones software no se pueden considerar como un evento inesperado, pero aún así se consideran excepciones por el tratamiento que se realiza de ellas
- El reset del microprocesador también se suele considerar como una excepción por coincidir su tratamiento con el de las excepciones en general

ARM7: Tratamiento de Excepciones

- Se pueden dividir las excepciones del ARM en tres grupos:
  - Excepciones generadas directamente por la ejecución de una instrucción
    - Interrupciones Software, SWI
    - Instrucciones no definidas
    - Prefetch aborts, debido a un error en el acceso a memoria al buscar un código de operación
  - Excepciones generadas como efectos secundarios en la ejecución de una instrucción
    - Data aborts, debido a un error en el acceso a memoria al buscar o almacenar un dato
  - Excepciones generadas externamente
    - Reset, IRQ, FIQ

ARM7: Tratamiento de Excepciones

- Proceso de tratamiento de una excepción
  - Todas las excepciones tienen el mismo tratamiento excepto la de reset
  - Cuando llega una excepción:
    - Se finaliza la ejecución de la instrucción en curso
    - La siguiente instrucción a ejecutar será la correspondiente a la rutina de la excepción asociada
    - Se cambia el modo de operación al modo correspondiente a la excepción que se produce (svc, undefined, abort, fiq, irq)
    - Se salva el contador de programa en el r14 del modo al que se ha cambiado
    - Se salva el CPSR en el SPSR correspondiente
    - Se desactivan las interrupciones IRQ siempre, y también las FIQ en caso de que esta sea la fuente de excepción
    - Se carga en el PC el vector correspondiente al modo en el que se entra

ARM7: Tratamiento de Excepciones

- Reset
  - Al activar el terminal de reset:
    - Los registros r14 y SPRS toman valores impredecibles
    - Se activa el modo supervisor
    - Se activa el modo ARM
    - Se inhabilitan las interrupciones IRQ y FIQ
    - Se carga en el PC el vector de reset

ARM7: Tratamiento de Excepciones

- Tabla de vectores de excepción

Exception	Mode	Vector address Normal	High
Reset	SVC	0x00000000	0xFFFF0000
Undefined instruction	UND	0x00000004	0xFFFF0004
Software interrupt (SWI)	SVC	0x00000008	0xFFFF0008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C	0xFFFF000C
Data abort (data access memory fault)	Abort	0x00000010	0xFFFF0010
IRQ (normal interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (fast interrupt)	FIQ	0x0000001C	0xFFFF001C

- En la dirección de cada vector se pone una instrucción de salto a la dirección de la rutina de tratamiento de la excepción, excepto en las interrupciones FIQ, que al ser el último vector, puede empezar el código de forma inmediata, con el consiguiente ahorro de tiempo

ARM7: Tratamiento de Excepciones

- Dentro de la rutina de tratamiento, se suelen almacenar en la pila particular los registros que se vayan a emplear, a excepción de las FIQ, que al tener duplicados los registros r8 a r12, puede hacer uso de ellos sin salvarlos previamente
  - Los punteros de pila se deben inicializar, en modo supervisor en el arranque
- En el retorno de las excepciones se debe:
  - Recuperar el contenido original de los registros empleados en la rutina
  - Restaurar el CPSR a partir del SPSR
  - Restaurar el valor del PC a su valor antes de la ejecución de la rutina
- Las dos últimas operaciones deben llevarse a cabo simultáneamente, de otro modo no sería posible reestablecer su contenido original

ARM7: Tratamiento de Excepciones

- Existen instrucciones que permiten realizar las dos operaciones simultáneamente, y que difieren según el modo en que se encuentre el micro
  - Retorno de una interrupción software o instrucción indefinida

```
MOVS    pc, r14          ; La s, asociada al pc, fuerza que se vuelque el contenido del
                         ; SPSR (SPSR_svc en este caso) al CPSR
```

  - Retorno de una interrupción (IRQ, FIQ), o una excepción "prefetch aborts"

```
SUBS    pc, r14, #4      ; Retoma el programa en la instrucción siguiente
```

  - Retorno de una excepción "data abort"

```
SUBS    pc, r14, #8      ; Esto permite volver a ejecutar la instrucción que dio lugar al error
```

  - El primer tipo no requiere modificación del contenido de r14 por ser excepciones esperadas, en cuyo caso se almacena en r14 el valor de la dirección de retorno

ARM7: Tratamiento de Excepciones

- Cada modo de funcionamiento está asociado a una excepción, menos el modo system
- Para acceder al modo system debe modificarse el CPSR desde un modo supervisor, ya que desde el modo user el registro de estado no se puede modificar

ARM7: Tratamiento de Excepciones

- Prioridad de las excepciones
  - Algunas de las excepciones existentes no pueden producirse simultáneamente, pero en el caso de varias se produjeran en el mismo instante, existen una prioridades preestablecidas tal como se muestra en la tabla siguiente

Prioridad	Excepción
1 Máxima Prioridad	Reset
2	Data Abort
3	FIQ
4	IRQ
5	Prefetch Abort
6 Mínima prioridad	Undefined instruction SWI

ARM7: Conjunto de Instrucciones Thumb

- El conjunto de instrucciones Thumb está diseñado para conseguir una mayor densidad de código
- Esta compuesto por un subconjunto de instrucciones del ARM comprimidas
- Para ejecutar una instrucción Thumb, se carga en el micro, se descomprime y se interpreta como una instrucción ARM estándar
- No se puede emplear el modo Thumb en el tratamiento de excepciones, cuando se produce una excepción, se conmuta de forma automática a modo ARM

ARM7: Conjunto de Instrucciones Thumb

- El modo de ejecución está definido por el bit T del registro CPSR
- Tras el reset, el micro arranca en modo ARM
- El paso a modo Thumb se realiza con una instrucción de salto BX, Branch and eXchage
- Se sale de modo Thumb ejecutando de nuevo la instrucción BX
- Las instrucciones Thumb ocupan 16 bits
- En este modo el acceso a los registros r8-r12 está restringido, solo es posible con determinadas instrucciones

ARM7: Conjunto de Instrucciones Thumb

- Para reducir el tamaño de las instrucciones
  - La mayoría de ellas no se pueden ejecutar condicionalmente
  - Se elimina uno de los operandos fuente y se sustituye por el operando destino
- Propiedades del modo de ejecución Thumb
  - El código requiere un 70% del espacio del código ARM
  - Usa un 40% más instrucciones que el código ARM
  - Con memorias de 32 bits, el código ARM es un 40% más rápido
  - Con memorias de 16 bits, el código Thumb es un 45% más rápido
  - Consume un 30% menos de potencia en acceso a memorias externas
- Conclusiones
  - Velocidad $\Rightarrow$ modo ARM, Bajo consumo, memoria de 16 bits $\Rightarrow$ modo Thumb

ARM7: Conjunto de Instrucciones Thumb

- Para reducir el tamaño de las instrucciones
  - La mayoría de ellas no se pueden ejecutar condicionalmente
  - Se elimina uno de los operandos fuente y se sustituye por el operando destino
- Propiedades del modo de ejecución Thumb
  - El código requiere un 70% del espacio del código ARM
  - Usa un 40% más instrucciones que el código ARM
  - Con memorias de 32 bits, el código ARM es un 40% más rápido
  - Con memorias de 16 bits, el código Thumb es un 45% más rápido
  - Consume un 30% menos de potencia en acceso a memorias externas
- Conclusiones
  - Velocidad $\Rightarrow$ modo ARM, Bajo consumo, memoria de 16 bits $\Rightarrow$ modo Thumb