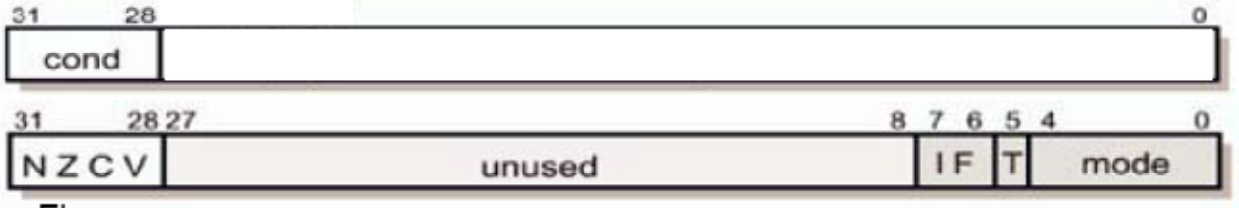
Sistemas electrónicos digitales

Tema 5:
El Juego de instrucciones del ARM



Flags

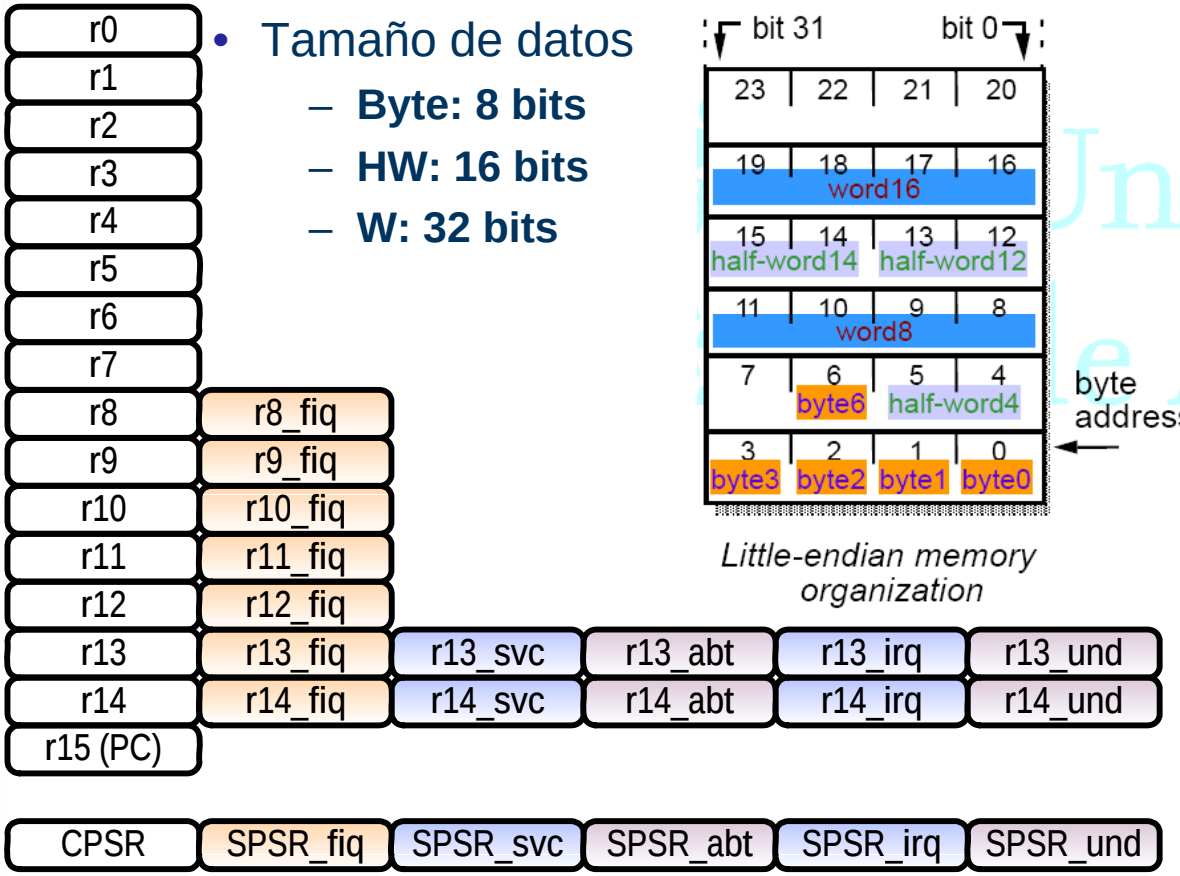
Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Zset
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	Cset

Índice del tema

- Introducción
- Excepciones en el ARM
- Codificación de instrucciones en el ARM
- Instrucciones de coprocesador
- Instrucción de punto de ruptura
- Espacio de instrucciones no utilizadas
- Fallos de memoria

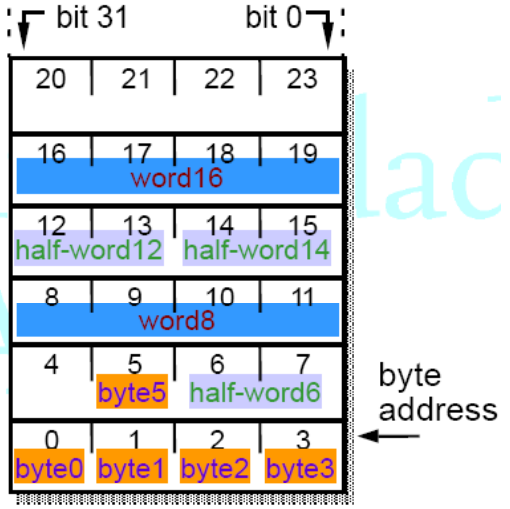
Introducción: Modelo de programación

- Objetivo: modos de excepción y de supervisor
- Recordatorio:



- Tamaño de datos
  - **Byte: 8 bits**
  - **HW: 16 bits**
  - **W: 32 bits**

Little-endian memory organization



Big-endian memory organization

Introducción: Modos privilegiados

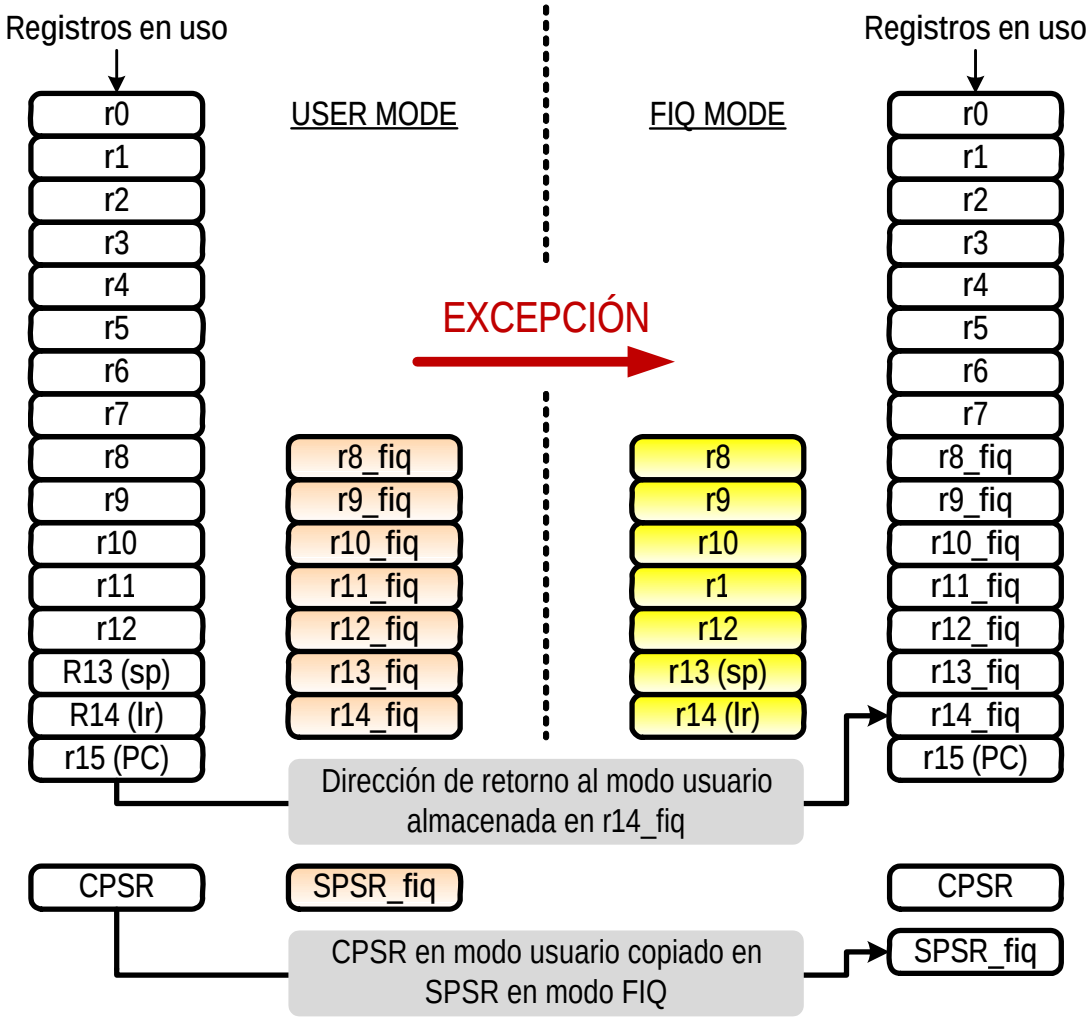
- La mayor parte de los programas se ejecutan en modo usuario, sin embargo se dispone de modos de operación **privilegiados**
  - Uso: manejo de excepciones y llamadas al supervisor (interrupciones por software)
  - Estos modos proporcionan un nivel básico de protección en sistemas empotrados
  - Cada modo, excepto el *system mode*, tiene su propio *Saved Program Status Register (SPSR)*

CPSR[4:0]	Modo	Reg.
10000	User	user
10001	FIQ	_fiq
10010	IRQ	_irq
10011	SVC	_svc

CPSR[4:0]	Modo	Reg.
10011	SVC	_svc
10111	Abort	_abt
11011	Undef	_und
11111	System	user

Introducción: Ejemplo de cambio de modo

Ejemplo del uso de los registros en modo FIQ



Excepciones en el ARM: Tabla de vectores de excepc.

Tabla de vectores de excepción:

Exception	Mode	Vector Normal	Vector High
Reset	svc	0x00000000	0xFFFF0000
Undefined instruction	und	0x00000004	0xFFFF0004
Software interrupt	svc	0x00000008	0xFFFF0008
Prefetch abort (instruction fetch mem. 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 debe localizar un salto a la dirección de comienzo de la Rutina de Servicio de la Excepción, excepto en el caso de FIQ que, al ser la última, puede localizarse el código de la RSE con el consiguiente ahorro de tiempo

Excepciones en el ARM: Procedimiento

Procedimiento de atención a una excepción

```
R14_<exception_mode> = return link
SPSR_<exception_mode> = CPSR
CPSR[4:0] = exception mode number
CPSR[5] = 0                 /* Execute in ARM state */
if <exception_mode> == Reset or FIQ then
      CPSR[6] = 1           /* Disable fast interrupts */
/* else CPSR[6] is unchanged */
      CPSR[7] = 1           /* Disable normal interrupts */
PC = exception vector address
```

Excepciones en el ARM: Prioridades

Prioridades en las excepciones en el ARM

- Mecanismo que decide la excepción más prioritaria en caso de que éstas aparezcan simultáneamente
- Escenario:
  - Recepción simultánea de *FIQ, IRQ* y una tercera excepción menos prioritaria
    - FIQ → IRQ → 3ª excepción
  - Recepción simultánea de *FIQ, IRQ* y *Data Abort*
    - *Data Abort* → FIQ → *Data Abort* → IRQ
  - Este procedimiento ocurre debido a que en la secuencia de acciones para atender a una interrupción se enmascara la interrupción IRQ, excepto en la atención a la interrupción FIQ, donde se enmascaran ambas

En orden de prioridades
Reset (excepción más prioritaria)
Data Abort
FIQ
IRQ
Prefetch Abort
SWI, Undefined Instruction: Éstas son mutuamente exclusivas al no poder aparecer simultáneamente

Excepciones en el ARM: *SWI*

Software Interrupt

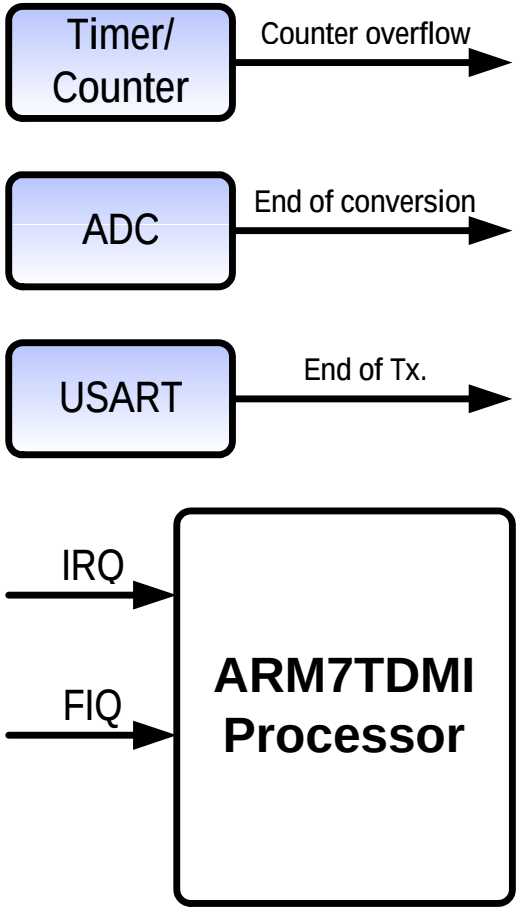
- Consiste en una instrucción definida por el usuario
- Su ejecución provoca una *exception trap*, al vector de interrupción de SWI, lo que provoca un cambio a modo supervisor y la ejecución de la RSE (*Interrupt Service Routine*)
- Sintaxis:

```
SWI     <Operation>          ; Se indica la operación a realizar
```

- Mediante el mecanismo de SWI se puede dotar al Sistema Operativo de una serie de funciones privilegiadas (principalmente de acceso a recursos *hardware*) que pueden ser invocadas por las aplicaciones (programas) que se ejecutan en modo usuario

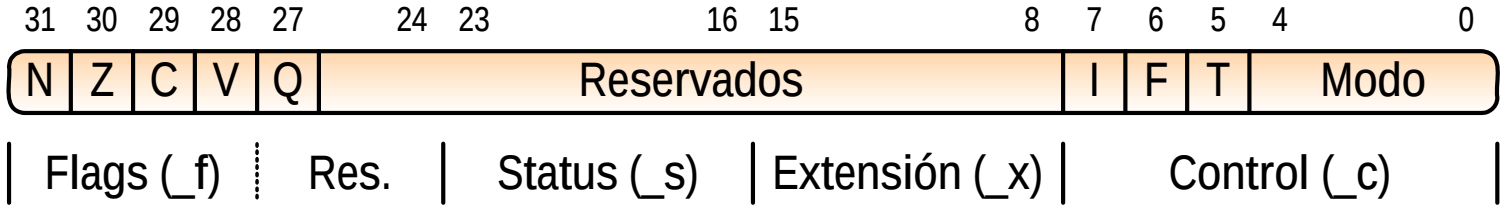
Excepciones en el ARM: Interrupc. por hardware I

- Hardware Interrupts
  - Internas
    - Generadas por periféricos internos o bien por software
  - Externas
    - *Normal Interrupt*, corresponde a la entrada IRQ del interfaz. Tiene una latencia de 25 ciclos
    - *Fast Interrupt*, corresponde a la entrada FIRQ del interfaz. Tiene una latencia de 12 ciclos.
- RSE de interrupciones externas
  - Borrar la fuente de interrupción antes de retornar al programa



Excepciones en el ARM: MRS/MSR I

- MRS: Mover el valor actual del CPSR/SPSR a un RPG
- MSR: Mover el valor de un RPG/dato inmediato al CPSR/SPSR
  - Permiten examinar y manipular el CPSR/SPSR mediante instrucciones aritméticas/lógicas y de comparación
  - Usadas normalmente para cambiar el modo y/o habilitar o deshabilitar las interrupciones (siempre en modo privilegiado)
  - El registro de estado está dividido en cuatro campos de 8 bis que pueden ser tratados de forma independiente



Excepciones en el ARM: MRS/MSR II

Sintaxis de MRS y MSR

```
MRS     Rd, CPSR  ; Copia del CPSR
MRS     Rd, SPSR          ; Copia del SPSR
MSR     CPSR_fields, Rm   ; Actualiza el CPSR desde un RPG
MSR     SPSR_fields, Rm   ; Actualiza el SPSR desde un RPG
MSR     CPSR_fields, #data ; Actualiza el CPSR desde un dato inmediato
MSR     SPSR_fields, #data ; Actualiza el SPSR desde un dato inmediato
```

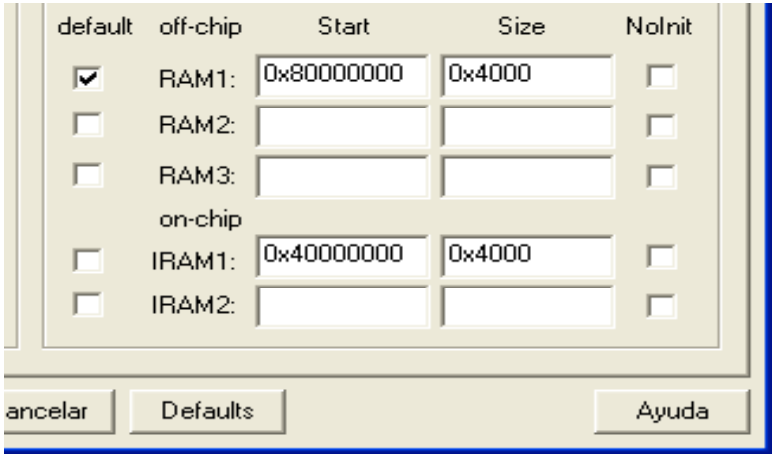
- *fields* puede ser cualquier combinación de “cxsf”

Ejemplo

```
MRS     R0, CPSR          ; Se lee el CPSR y se copia en R0
BIC     R0, R0, #0x1F     ; Se borran los bits de Modo
ORR     R0, R0, #0x13     ; Se actualizan a Modo Supervisor
MSR     CPSR_c, R0        ; Se escribe el nuevo valor en CPSR
```

Excepciones en el ARM: Ejemplo con Keil DT I

- Ejemplo de tratamiento de una excepción *Data Abort*
  - *Keil DT → Target 2114*
    - Deshabilitamos la memoria RAM interna y localizamos memoria RAM *off-chip*



  - Acciones
    - R1 ← 0x80000000
    - Excepción por *Data Abort* (vector 0x10)
      - Es necesario configurar el entorno para usar memoria externa

```
07              AREA    MAIN, CODE, READONLY
08              EXPORT  __main, MyDAbt_Handler
09
10              ENTRY
11  __main      ADR     r0, ENTRADA
12  ; SALIDA localizada en RAM
13              LDR     r1, =SALIDA
14  ; Copiamos la Entrada en la Salida
15              MOV     r3, #0
16  LOOP0       LDRB    r2, [r0], #1
17              STRB    r2, [r1], #1
```

```
Load "D:\\Mis asignaturas\\Sistemas Electronicos Digitales ARI
Data Abort: ARM Instruction at 00000120H, Memory Access at 80(
```

Excepciones en el ARM: Ejemplo con Keil DT II

- Salto al vector 0x10 para el manejo de la ISR correspondiente

```
LDR     PC, DAbt_Handler
```

- *Keil Startup.s for LPC21xx*
  - Define una posición de memoria conteniendo la dirección de comienzo de la ISR
  - Esta dirección se carga en PC, lo que provoca un salto a la ISR
- Posibilidad de definir nuestras propias rutinas de servicio

Startup.s

```
232 Vectors         LDR     PC, Reset_Addr
233                 LDR     PC, Undef_Addr
234                 LDR     PC, SWI_Addr
235                 LDR     PC, PAbt_Addr
236                 LDR     PC, DAbt_Addr
237                 NOP
238 ;               LDR     PC, IRQ_Addr
239                 LDR     PC, [PC, #-0x0FF0]
240                 LDR     PC, FIQ_Addr
241
242 Reset_Addr     DCD     Reset_Handler
243 Undef_Addr     DCD     Undef_Handler
244 SWI_Addr       DCD     SWI_Handler
245 PAbt_Addr      DCD     PAbt_Handler
246 DAbt_Addr      DCD     DAbt_Handler
247                 DCD     0
248 IRQ_Addr       DCD     IRQ_Handler
249 FIQ_Addr       DCD     FIQ_Handler
250
251 Undef_Handler  B       Undef_Handler
252 SWI_Handler    B       SWI_Handler
253 PAbt_Handler   B       PAbt_Handler
254 DAbt_Handler   B       DAbt_Handler
255 IRQ_Handler    B       IRQ_Handler
256 FIQ_Handler    B       FIQ_Handler
```

Excepciones en el ARM: Definición y uso de una ISR

- Definición de ISR propias
  - Modificación de la dirección de salto mediante la redefinición del contenido de *DAbt_Addr*
  - Modificación del contenido del vector de interrupción

```
LDR     PC, PAbt_Addr
LDR     PC, MyDAbt_Handler
NOP
LDR     PC, IRQ_Addr
LDR     PC, [PC, #-0x0FF0]
```

  - Error de concepto en el uso de LDR para saltar a la ISR
  - Solución: uso de

```
LDR  Rn, =val
```

```
LDR     PC, PAbt_Addr
LDR     PC, =MyDAbt_Handler
```

Startup.s

```
231 Vectors         LDR     PC, Reset_Addr
232                 LDR     PC, Undef_Addr
233                 LDR     PC, SWI_Addr
234                 LDR     PC, PAbt_Addr
235                 LDR     PC, DAbt_Addr
236                 NOP
237 ;               LDR     PC, IRQ_Addr
238                 LDR     PC, [PC, #-0x0FF0]
239                 LDR     PC, FIQ_Addr
240
241                 EXTERN  MyDAbt_Handler
242 Reset_Addr      DCD     Reset_Handler
243 Undef_Addr      DCD     Undef_Handler
244 SWI_Addr        DCD     SWI_Handler
245 PAbt_Addr       DCD     PAbt_Handler
246 DAbt_Addr       DCD     MyDAbt_Handler
247                 DCD     0
248 IRQ_Addr        DCD     IRQ_Handler
249 FIQ_Addr        DCD     FIQ_Handler
```

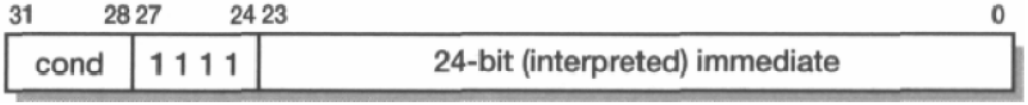
Exc_handler.s

```
40          AREA    EXC_HAND, CODE, READONLY
41          EXPORT  MyDAbt_Handler
42
43 MyDAbt_Handler  B       MyDAbt_Handler
```

Excepciones en el ARM: Uso de SWI I

- Utilizada para llamadas al sistema operativo
- Posibilidad de crear rutinas propias mediante el manejo de SWI. Gestión con un único vector de interrupción
  - Formato y codificación

```
SWI     IntNumber
```



  - `IntNumber` codifica la función a ejecutar dentro de la ISR
- Acciones en la *SWI ISR*
  - 1. Comprobación de modo: ARM o *Thumb*
  - 2. Extraer el número de la función a ejecutar
  - 3. Comprobar que la función existe
  - 4. Ejecutar la función correspondiente
  - 5. Retornar al programa principal

Excepciones en el ARM: Uso de SWI II

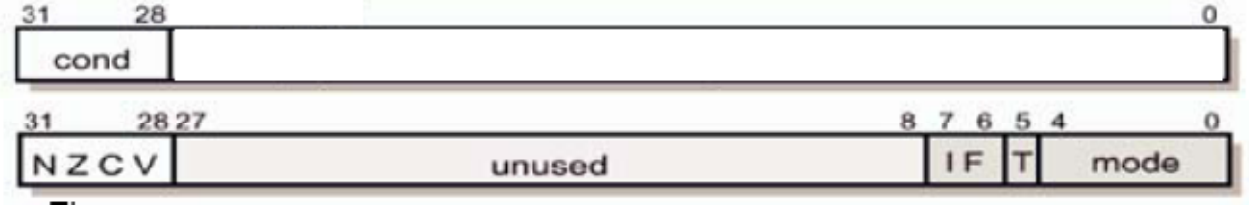
```
14  SWI_Handler?A    PROC CODE32
15          STMFD    SP!,{LR}              ; Store LR register
16  1       MRS      R8,SPSR
17          TST      R8,#0x20              ; SWI call from
18          LDRNEH   R8,[LR,#-2]           ; Thumb: Load halfword instruction
19          ANDNE    R8,R8,#0xFF           ;        extract SWI number
20  2       LDREQ    R8,[LR,#-4]           ; ARM:   Load word     instruction
21          BICEQ    R8,R8,#0xFF000000     ;        extract SWI number
22                                         ; R4 now contains SWI number
23  ; SWI Handler
24          ADR      R12,?SWI?Table
25          LDR      R12,[R12]             ; load last SWI-Function-number
26  3
27          CMP      R8,R12
28          BGT      ?SWI?Empty            ; overflow
29          ADR      R12,?SWI?Table+4
30          LDR      R12,[R12,R8,LSL #2]   ; SWI function address
31  4       MOV      LR,PC                 ; Return address
32          BX       R12                   ; Call SWI function
33  5       LDMFD    SP!,{PC}^             ; Return
34
35  ?SWI?Empty:
36          B        $                     ; no existing SWI
37  ; *** DO NOT MODIFY THIS PORTION OF THE FILE ***
38  ?SWI?Table:                            ; Marker for LA Linker
39  ;
40  ; The LA Linker inserts at this label
41  ;       DD       0                     ; <last SWI function number>
42  ;       DD       ?SWI?Empty            ; <entry for SWI function 0>
43  ;       DD       <SWI entry 1>
44  ;       DD       <SWI entry 2>
45  ;       DD            :
```

Codificación de instrucciones en el ARM: Introducción

- Todas las instrucciones del ARM son de 32 bits y están alineadas a nivel de palabra, *4-byte boundaries*
- Algunas versiones soportan la ejecución de instrucciones en modo comprimido, *Thumb mode*
- Únicamente las operaciones de transferencia soportan operandos de tamaño corto, *byte* y *halfword*
- Internamente, todas las operaciones se realizan sobre operandos de 32 bits. La transferencia de un dato corto a registro, implica su extensión a 32 bits

Codificación de instrucciones en el ARM: Ejecución condicional

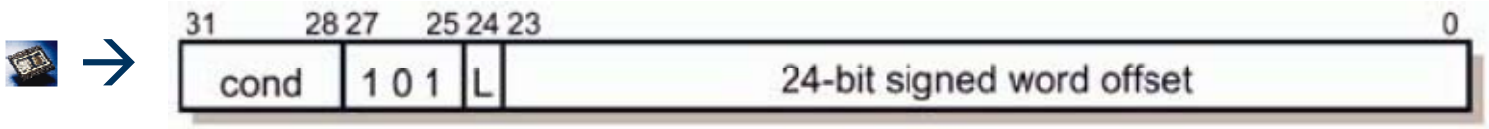
- Todas las instrucciones del ARM tienen la posibilidad de ejecución condicional
  - El campo de condición de la instrucción ocupa los cuatro bits más significativo de su codificación
  - La codificación de la condición por la cual se ejecuta/salta una instrucción, se basa en el estado de los bits de condición del registro de estado CPSR



Flags

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Zset
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	Cset
1110	AL	Always	any
1111	NV	Never (do not use!)	none

Codificación de instrucciones en el ARM: Branch and Branch and Link



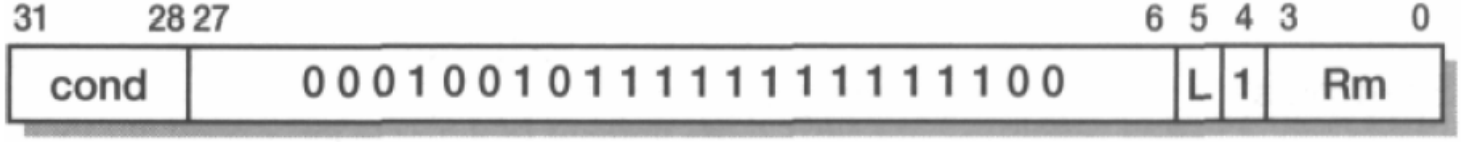

```
B{L}{cond}          <target_address>
```

- Si no se especifica la condición, se presupone “AL” (Always)
- Bits[27:25]: Opcode, Branch or Branch and Link
- Bit[24] (L): Distingue entre Branch (L=0) o BL (L=1)
- Bits[23:0]: *Signed Offset* referente a la dirección de salto
- El ensamblador calcula la dirección efectiva mediante
  - EA ← PC + 8 + *Offset* << *2*
- En caso de ser BL, la dirección de retorno queda almacenada en r14
- El retorno de subrutina se consigue copiando el valor de r14 en el PC

```
; Ejecución condicional
CMP     r0, #5 ; If r0 < 5
BLLT    SUB1   ; Then call SUB1
BLGE    SUB2   ; Else call SUB2
```

Codificación de instrucciones en el ARM: Branch and Branch and Link and excahnge I

- Disponibles únicamente en procesadores con soporte *Thumb*
- Permiten conmutar entre los modos *Thumb* y ARM en las llamadas y retorno de subrutina



BX/BLX Rm

- Si Rm[0] = 1 → Se pasa a modo *Thumb* y se continua la ejecución en la dirección especificada en Rm y alienada en *halfword*
- Si Rm[0] = 0 → Se continua en modo ARM y se continua la ejecución en la dirección especificada en Rm y alineada en *word*

BLX Label EA ← PC + 8 + Label << 2 + H << 1

- Label: desplazamiento de 24 bits con signo. El bit H posibilita el salto a a direcciones impares a nivel de *halfword*
- Siempre se produce el paso a modo *Thumb*

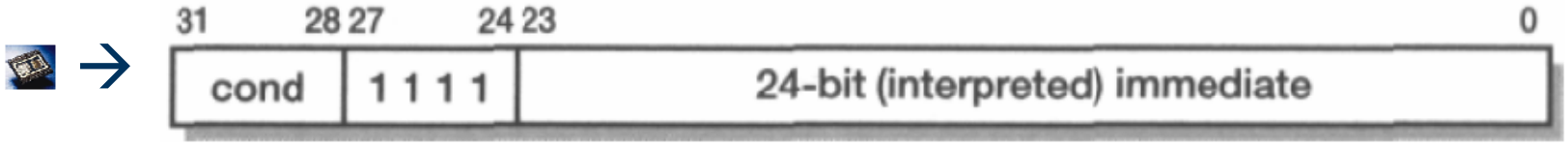
Codificación de instrucciones en el ARM: Branch and Branch and Link and excahnge II

- Posibilidad de implementar retornos de subrutina

```
        BL      SUB1            ; Llamada a SUB1
        ...                     ; Punto de retorno de SUB1
SUB1    STMFD   r13!, {r0-r2}   ; Salvar el estado de la máquina
        ...                     ; Cuerpo de la rutina
        LDMFD   r13!, {r0-r2}   ; Recuperar el estado
        BX      r14             ; Retorno de SUB1
```

  - En los procesadores que no soportan el modo *Thumb*, estas instrucciones pueden ser emuladas mediante excepciones tipo trap
  - La instrucción BLX está disponible únicamente en los procesadores que soportan la versión v5T

Codificación de instrucciones en el ARM: SWI



SWI{cond} <24-bit immediate>

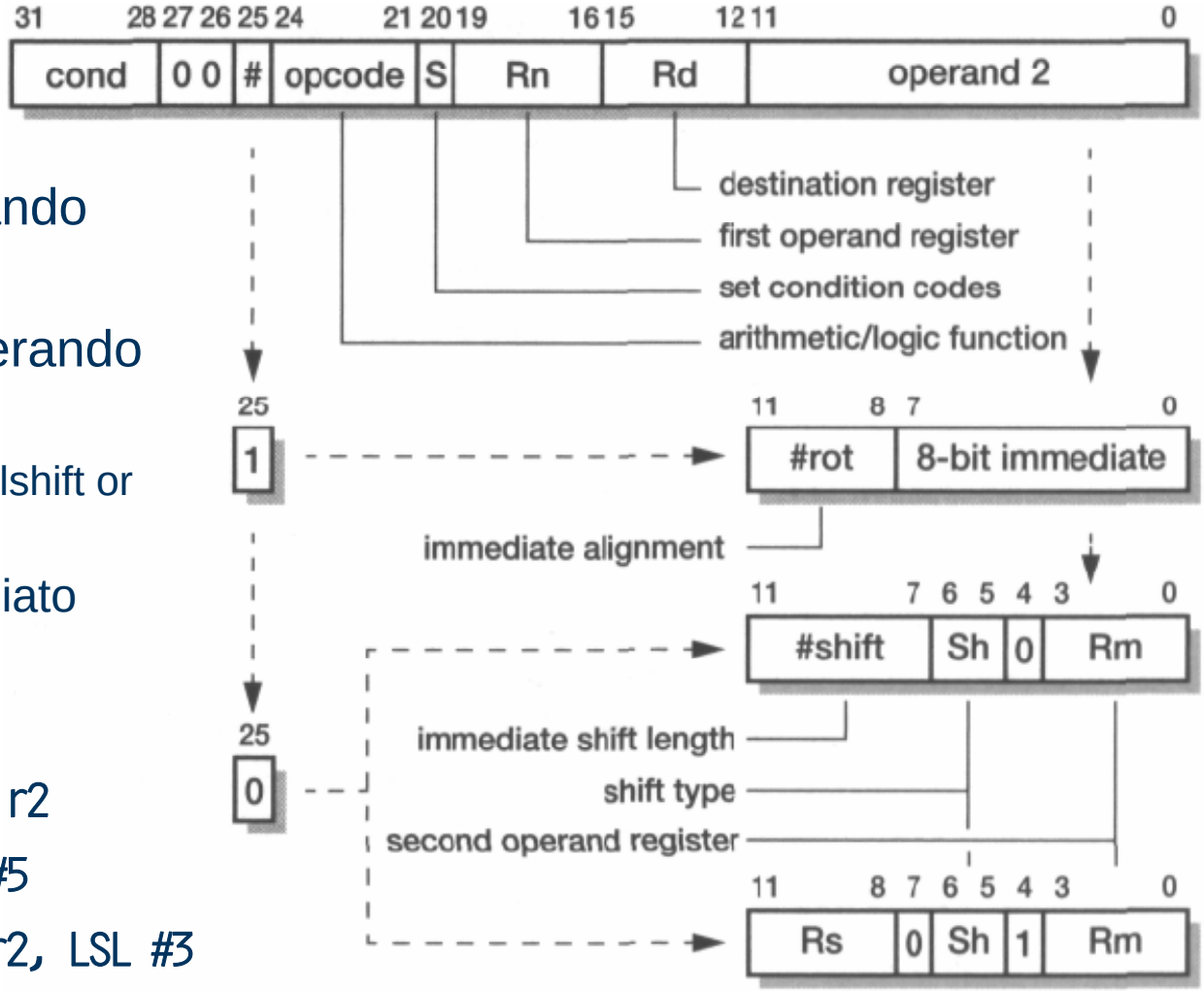
- Acciones
  - Salva la dirección de retorno en el registro r14_svc
  - Salva el CPSR en el SPSR_svc
  - Entra en modo supervisor y deshabilita la interrupción IRQ, aunque no la FIQ poniendo el valor %10011 en CPSR[4:0] y el bit CPSR[7] a “1”
  - Carga la dirección 0x08 en el PC para comenzar a ejecutar la ISR
  - La ejecución de SWI en modo supervisor requiere haber salvado previamente los valores de r14_svc y SPSR_svc

Codificación de instrucciones en el ARM: Instrucciones de procesamiento de datos I

Codificación

- Primer operando
  - registro
- Segundo operando
  - Registro
    - ashift, lshift or rotate
  - Op. Inmediato
    - rotate
- Ejemplos:

```
ADDS r0, r1, r2
ADD r0, r1, #5
ADD r0, r1, r2, LSL #3
ADD r0, r1, r2, LSL r3
```

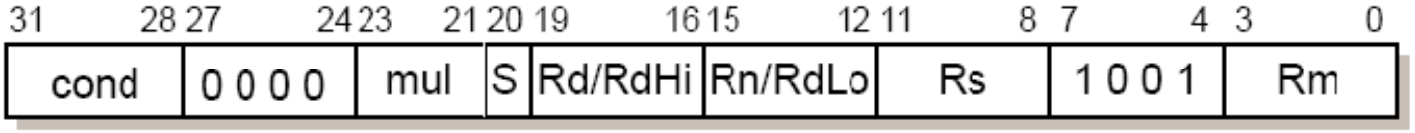


Codificación de instrucciones en el ARM: Instrucciones de procesamiento de datos II

Instrucciones

Opcode 124:21)	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd:=RnANDOp2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	SccconRnANDOp2
1001	TEQ	Test equivalence	Sec on Rn EOR Op2
1010	CMP	Compare	Sec on Rn - Op2
1011	CMN	Compare negated	Sec on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd:=RnANDNOTOp2
1111	MVN	Move negated	Rd:=NOTOp2

Codificación de instrucciones en el ARM: Instrucciones de multiplicación I




```
MUL{cond}{S}        Rd, Rm, Rs          ; 32-bits result
MLA{cond}{S}        Rd, Rm, Rs, Rn      ; 32-bits result
<mul>{cond}{S}      RdHigh, RdLow, Rm, Rs ; 64-bits result
```

- *Set conditions*: N; Z; C ← X; V ← No se modifica
- Multiplicación vs. Multiplicación y acumulación
- Multiplicación con resultado en 32-bits/64-bits (esta última no disponible en todos las versiones)

Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit result)	Rd:=(Rm*Rs)[31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd:=(Rm*Rs + Rn)[31:0]
100	UMULL	Unsigned multiply long	RdHirRdLo := Rm * Rs
101	UMLAL	Unsigned multiply-accumulate long	RdHi: RdLo +– Rm * Rs
110	SMULL	Signed multiply long	RdHi:RdLo := Rm * Rs
111	SMLAL	Signed multiply-accumulate long	RdHi:RdLo+=Rm*Rs

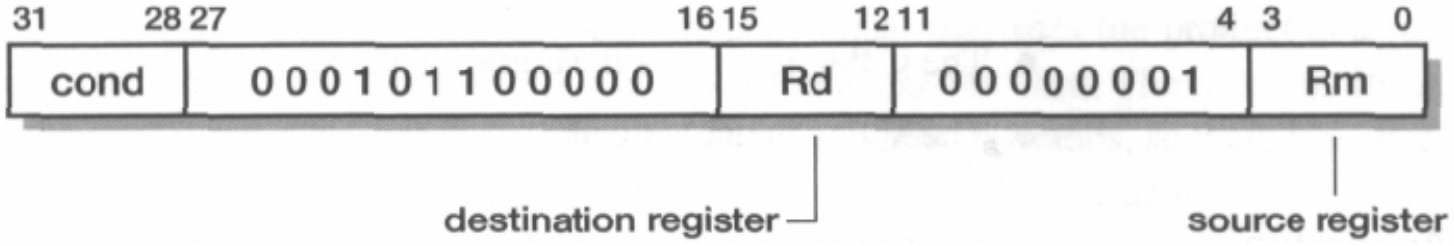
Codificación de instrucciones en el ARM: Instrucciones de multiplicación II

- Ejemplo: Producto escalar de dos vectores

```
        MOV     r11, #20            ; Inicializar contador
        MOV     r10, #0             ; r10 ← Suma parcial
LOOP    LDR     r0, [r8], #4        ; Acceso a un componente
        LDR     r1, [r9], #4        ; Acceso a un componente
        MLA     r10, r0, r1, r10    ; Multiplicación y acumul.
        SUBS    r11, r11, #1        ; Decremento del contador
        BNE     LOOP                ; Saltamos al bucle
```

  - No tiene sentido usar el PC como operando y menos como resultado
  - Rd, RdHi, RdLo deben ser distintos de Rm
  - RdHi y RdLo no deberían ser el mismo registro
  - Las versiones del ARM que soportan la multiplicación con 64 bits, se distinguen por la variante “M”, e.g. ARM7DM, ARM7TM

Codificación de instrucciones en el ARM: *Count Leading Zeros*



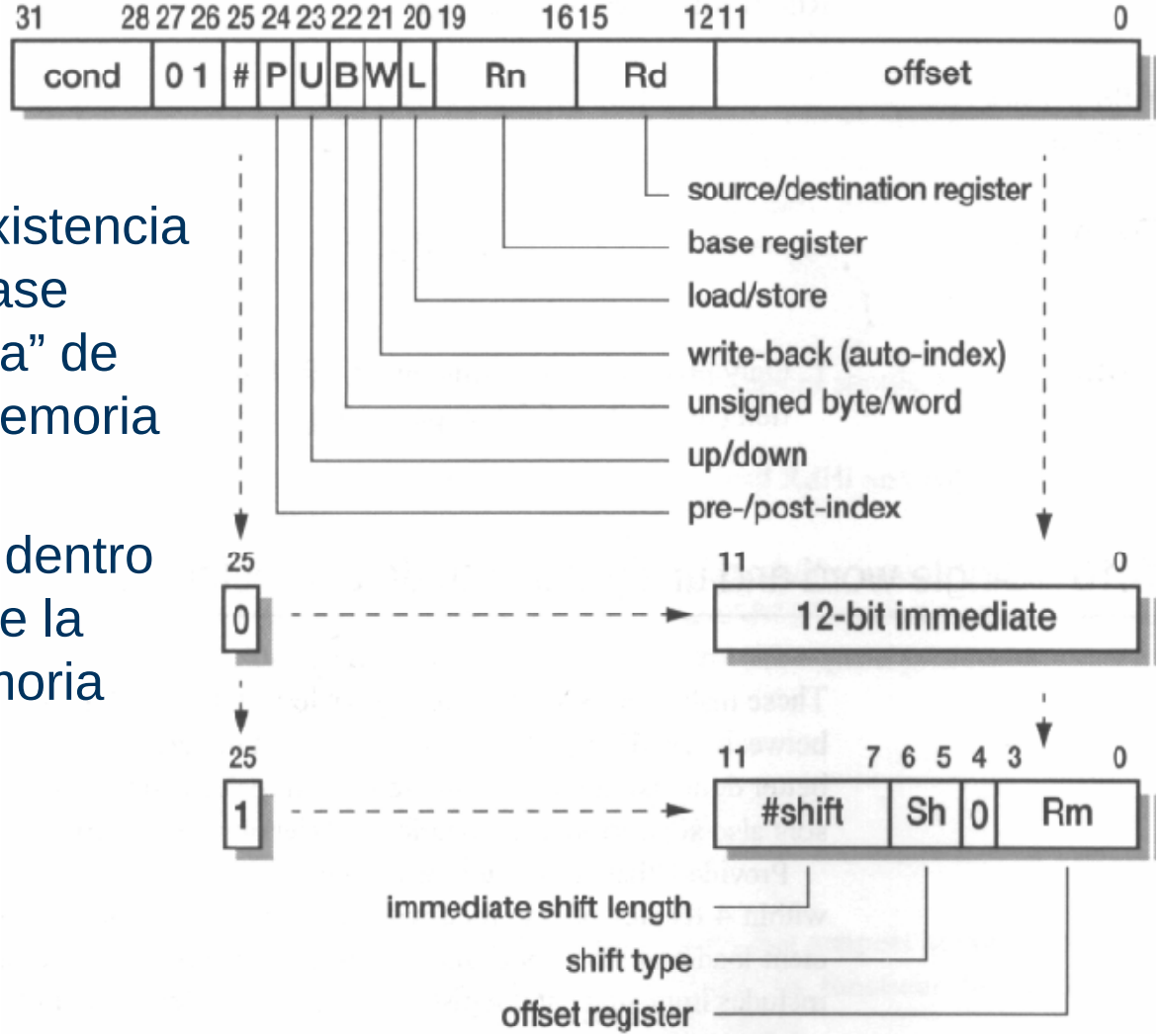

```
CLZ{cond}        Rd, Rm
```

- Disponible únicamente en procesadores que soportan la arquitectura v5T
  - Útil en procesamiento de datos en punto flotante para la normalización del operando
  - Rd ← Se carga con la posición del “1” más significativo en Rm
  - En caso de que Rm = 0, se carga el valor 32
  - Ejemplo

```
MOV     r0, #0x100      ; r0 ← 0x100
CLZ     r1, r0          ; r1 ← 8
```

Codificación de instrucciones en el ARM: Transferencia de datos, *single word & unsigned byte* I

- Codificación
- Anotaciones
  - Se basa en la existencia de un registro base apuntando “cerca” de la posición de memoria seleccionada
  - “cerca” posición dentro de los 4kbytes de la posición de memoria deseada

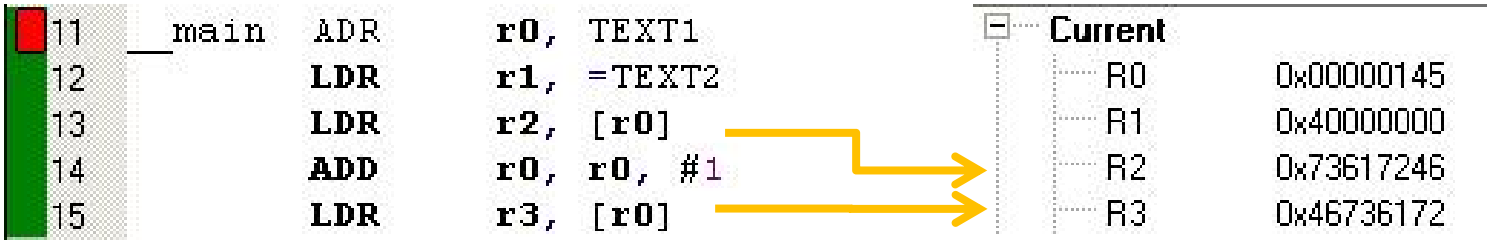


Codificación de instrucciones en el ARM: Transferencia de datos, *single word & unsigned byte* II

Formato

```
LDR/STR{cond}{B}          Rd, [Rn, offset]{!}
LDR/STR{cond}{B}{T}       Rd, [Rn], offset
LDR/STR{cond}{B}          Rd, label
```

- LDRT: *Load Register with Translation*
  - Si la instrucción se ejecuta en modo privilegiado, el acceso a memoria se realiza como si se tratara de modo usuario
- LDR: Acceso a palabras no alineadas



  - Addr[1:0] = 0b00 → Rd = Mem[addr,4]
  - Addr[1:0] = 0b01 → Rd = Mem[addr,4] rotate_right 8
  - Addr[1:0] = 0b10 → Rd = Mem[addr,4] rotate_right 16
  - Addr[1:0] = 0b11 → Rd = Mem[addr,4] rotate_right 24

Codificación de instrucciones en el ARM: Transferencia de datos, *single word & unsigned byte* III

- STR: Escritura de palabras no alineadas
  - Independientemente de la dirección de escritura, ésta se realiza de forma alineada en palabra

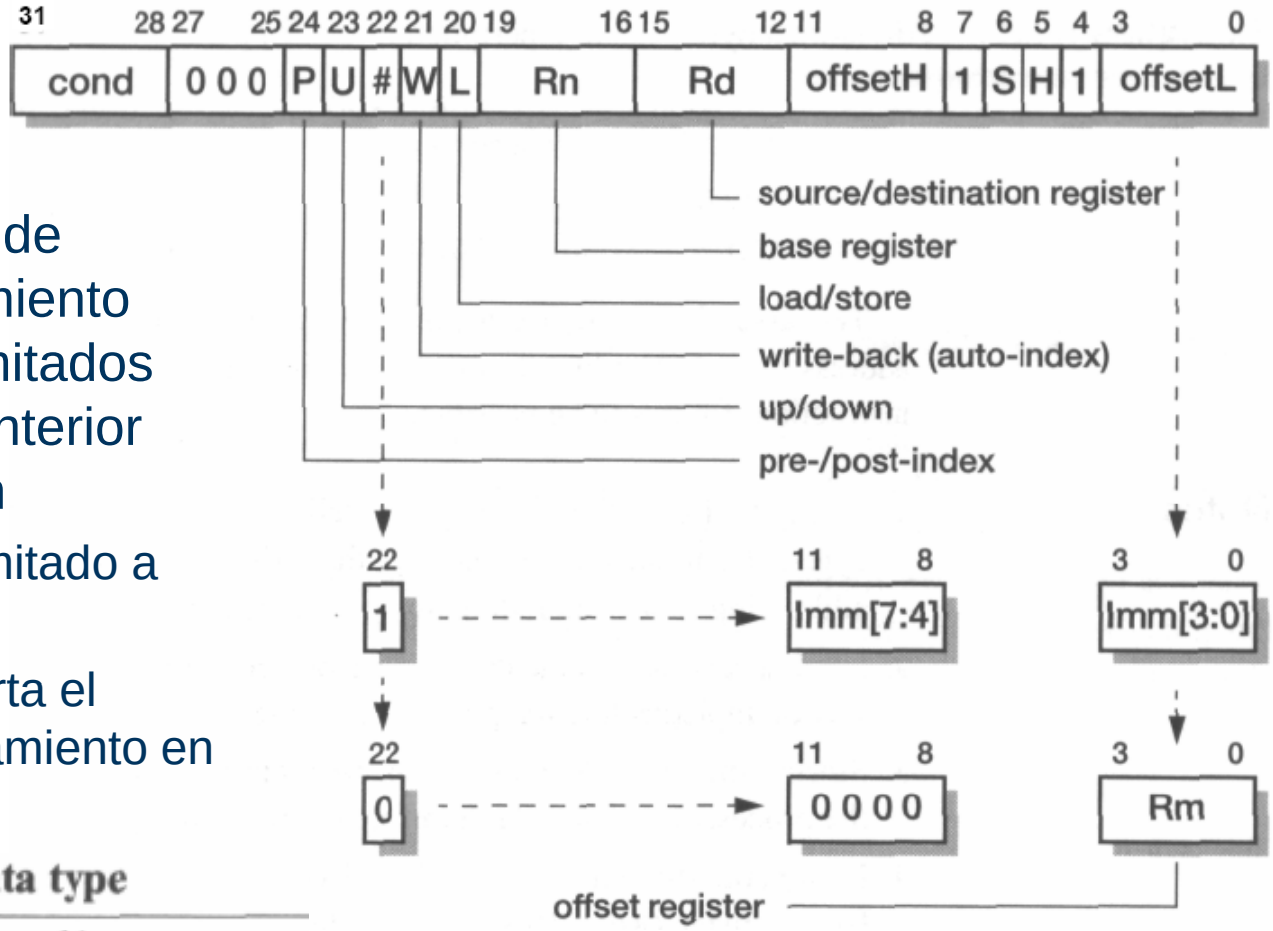
```
11  __main  ADR     r0, TEXT1
12          LDR     r1, =TEXT2
13          LDR     r2, [r0]
14          STR     r2, [r1]
15          ADD     r1, r1, #5
16          STR     r2, [r1]
```

Address: 0x40000000

```
0x40000000: 46 72 61 73 46 72 61 73 00 00
0x40000015: 00 00 00 00 00 00 00 00 00 00
0x4000002A: 00 00 00 00 00 00 00 00 00 00
```

Codificación de instrucciones en el ARM: Transferencia de datos, *half word & signed byte* I

- Codificación
- Anotaciones
  - Los modos de direccionamiento son más limitados que en la anterior codificación
    - *Offset* limitado a 8 bits
    - No soporta el desplazamiento en registros



S	H	Data type
1	0	Signed byte
0	1	Unsigned half-word
1	1	Signed half-word

Codificación de instrucciones en el ARM: Transferencia de datos, *half word & signed byte* II

Formato

```
LDR/STR{cond}H/SH/SB      Rd, [Rn, offset]{!}
LDR/STR{cond} H/SH/SB     Rd, [Rn], offset
```

- Acceso a palabras no alineadas: las direcciones siempre deben estar alineadas en *halfword*

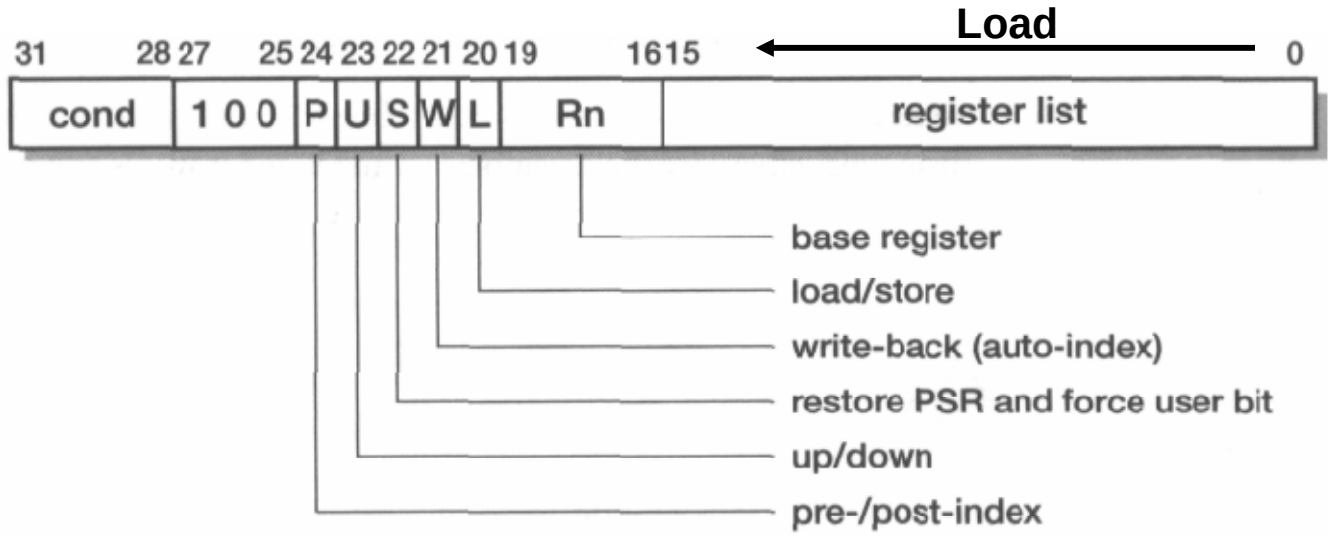
```
11  __main  ADR     r0, TEXT1
12          LDR     r1, =TEXT2
13          LDR     r2, [r0]
14          LDRH    r3, [r0]
15          LDRSH   r4, [r0]
16          LDRB    r5, [r0]
17          LDRSB   r6, [r0]
18          ADD     r0, r0, #1
19          LDRH    r7, [r0]
20          LDRSH   r8, [r0]
21          LDRB    r9, [r0]
22          LDRSB   r10, [r0]
```

Current

R0	0x0000016d
R1	0x40000000
R2	0x73619296
R3	0x00009296
R4	0xffff9296
R5	0x00000096
R6	0xffffff96
R7	0x00009296
R8	0xffff9296
R9	0x00000092
R10	0xffffff92

```
BS \main\20, 1
Non-aligned Access: ARM Instruction at 00000130H, Memory Access at 00000
Non-aligned Access: ARM Instruction at 00000134H, Memory Access at 00000
```

Codificación de instrucciones en el ARM: Transferencia múltiple de datos I




```
LDM/STM{cond}<add_mode>    Rn{!}, <registers +/-pc>
```

- La lista de registros no puede estar vacía
- Cada bit de la lista corresponde a un registro
- Uso de “pc” en la lista de registros
  - Si se añade a la lista de registros, y el bit S está a “1”, se recupera el valor del SPSR del modo actual sobre el CPSR. Se debe evitar su uso en modo usuario/*system* al no disponer de SPSR

Codificación de instrucciones en el ARM: Transferencia múltiple de datos II

- Usos de LDM/STM en modos no usuario. Uso del sufijo "^" al cerrar la lista de registros
  - Recuperación de CPSR

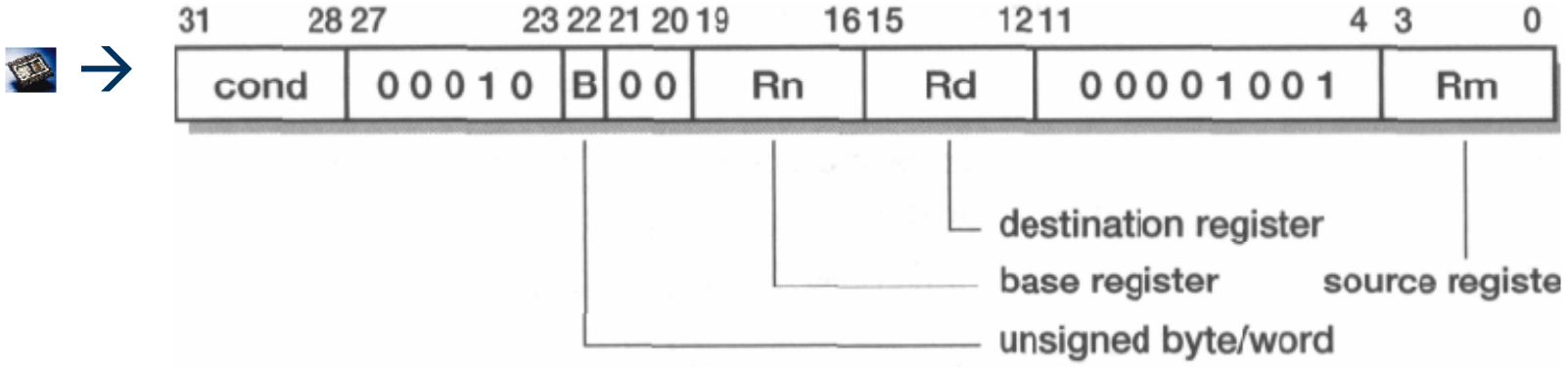
    `LDM{cond}<add_mode>  Rn{!}, <registers + pc>^`

    - En este caso se recupera el SPSR sobre el registro CPSR
    - El "pc" debe formar parte de la lista de registros
  - Posibilidad de salvar y recuperar los registros utilizados en modo usuario

    `LDM/STM{cond}<add_mode>  Rn, <registers - pc>^`

    - El "pc" no puede formar parte de la lista de registros
    - No se permite autoindexación
    - Esta característica en de mucha utilidad en la gestión de un sistema operativo

Codificación de instrucciones en el ARM: SWAP

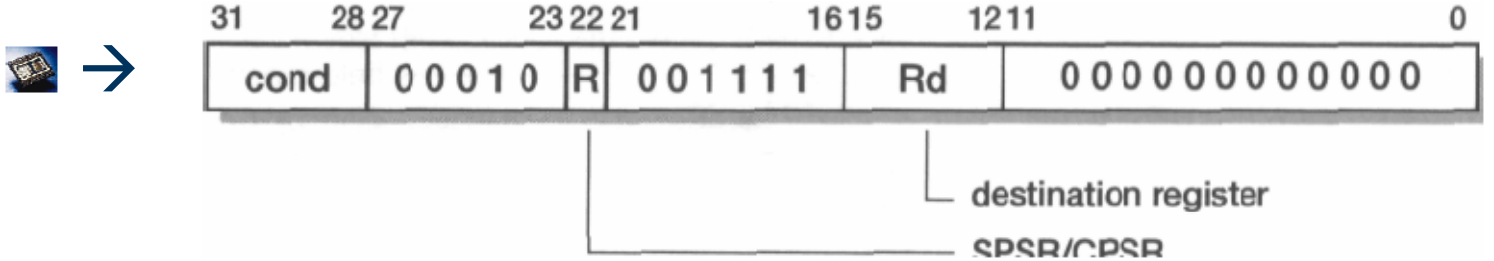



```
SWP{cond}{B}      Rd, Rm, [Rn]
```

- Combina un *load/store* de palabra o byte sin signo en una única instrucción
  - Rd ← [Rn]; [Rn] ← Rm
- Útil en la implementación de semáforos para el acceso a recursos compartidos

```
ADR     r0, SEMAPHORE
SWPB    r1, r1, [r0]      ; Intercambia byte
```

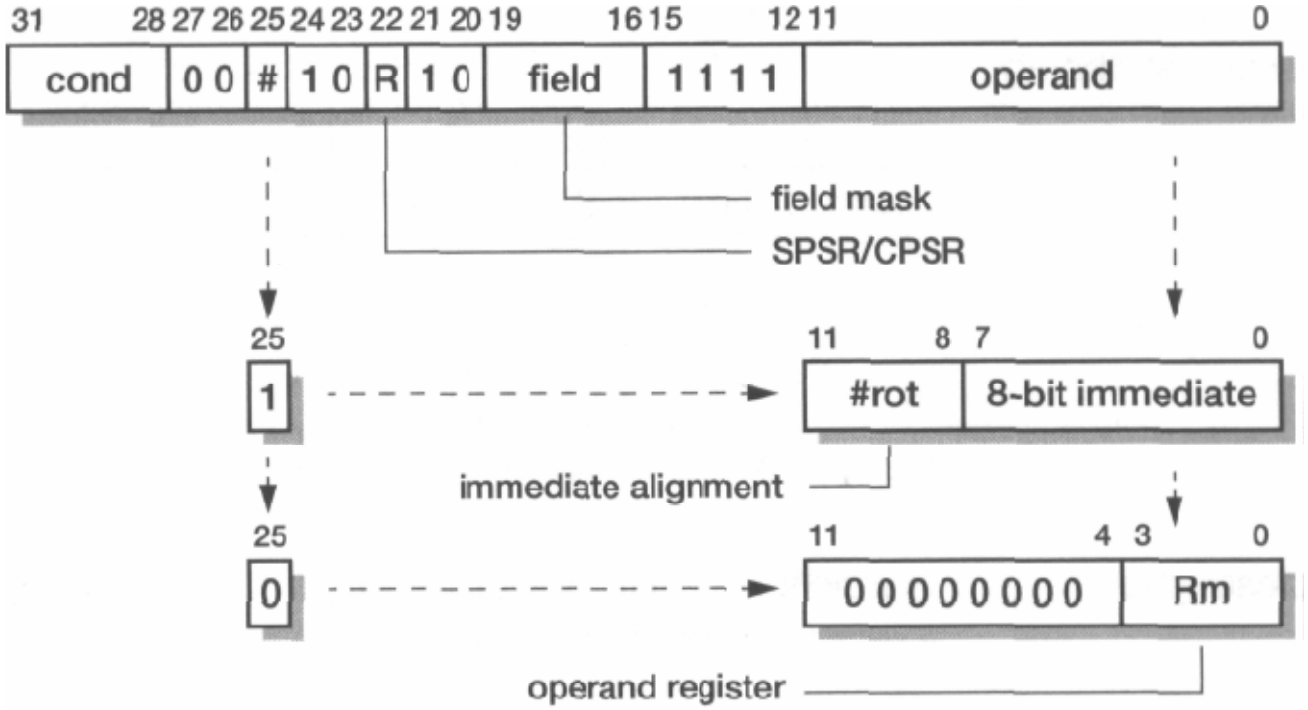
Codificación de instrucciones en el ARM: MRS



MRS{cond} Rd, CPSR/SPSR

- Evitar el uso de SPSR en modo usuario
- Se recomienda no modificar los bits no usados para prevenir futuras compatibilidades con nuevas versiones del ARM

Codificación de instrucciones en el ARM: MSR




```
MSR{cond}        CPSR_f/SPSR_f, #<32-bit immediate>
MSR{cond}        CPSR_fields/SPSR_fields, Rm
```

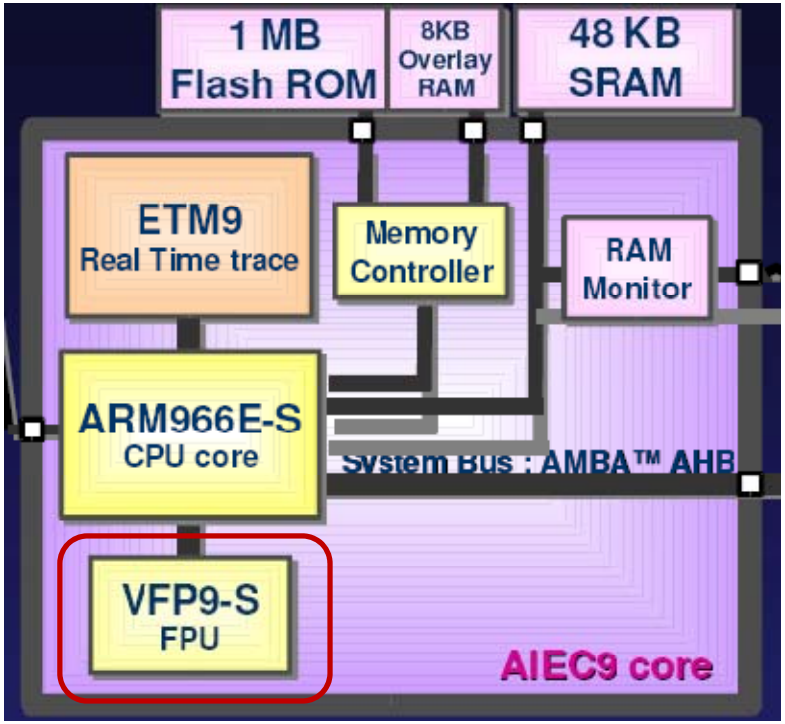
- Evitar el uso de SPSR en modo usuario

Instrucciones de coprocesador: Introducción

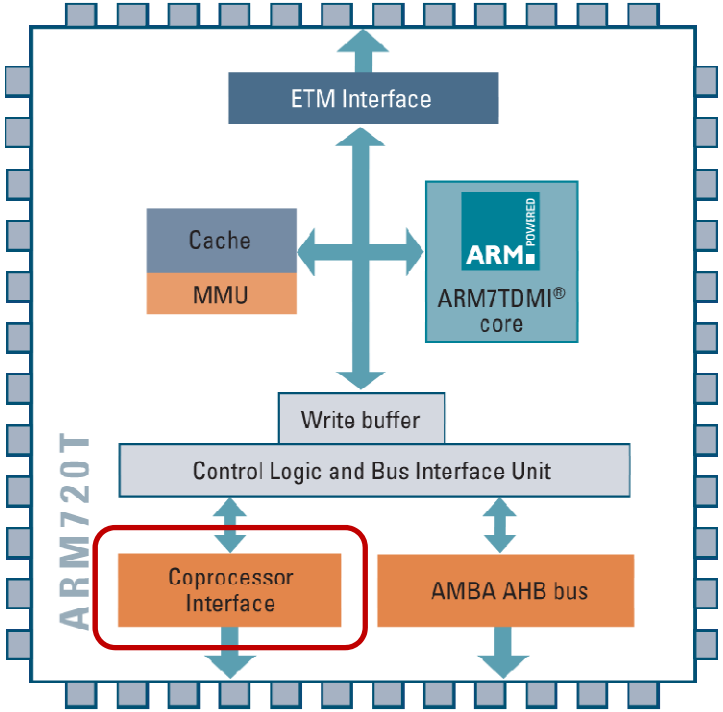
- Mecanismo para extender el conjunto de instrucciones mediante el uso de coprocesadores
- Los coprocesadores disponen de sus propios registros y siguen la filosofía de una arquitectura *load/store*
- El procesamiento de datos es interno y el resultado queda en un “tercer” registro
- Las transferencias se gestionan por ambas partes
  - El ARM genera la dirección de acceso
  - El coprocesador gestiona la longitud de la transferencia
- Existe la posibilidad de transferencia entre registros del coprocesador y del ARM

Instrucciones de coprocesador: Ejemplos de coprocesador

- VFP9. *Vector Floating Point Coprocessor*
  - Compatible con ARM9
  - Aplicaciones en la industria del automóvil y de control

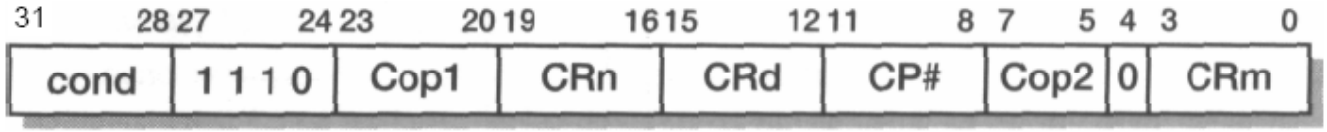


- CP15. Coprocesador para el control interno del dispositivo
  - Integrado en el núcleo del ARM720T



Instrucciones de coprocesador: *Data operations I*

- Se usan para iniciar una operación en el coprocesador
- La operación se realiza internamente, ejemplo
  - Multiplicación en punto flotante, que multiplica el contenido de dos registros y almacena el resultado en un tercer registro
  - Si es aceptada por un coprocesador el ARM pasa a la siguiente instrucción, en caso contrario se emula por instrucción no definida. En caso de estar presente el coprocesador especificado, éste acepta la instrucción
- →

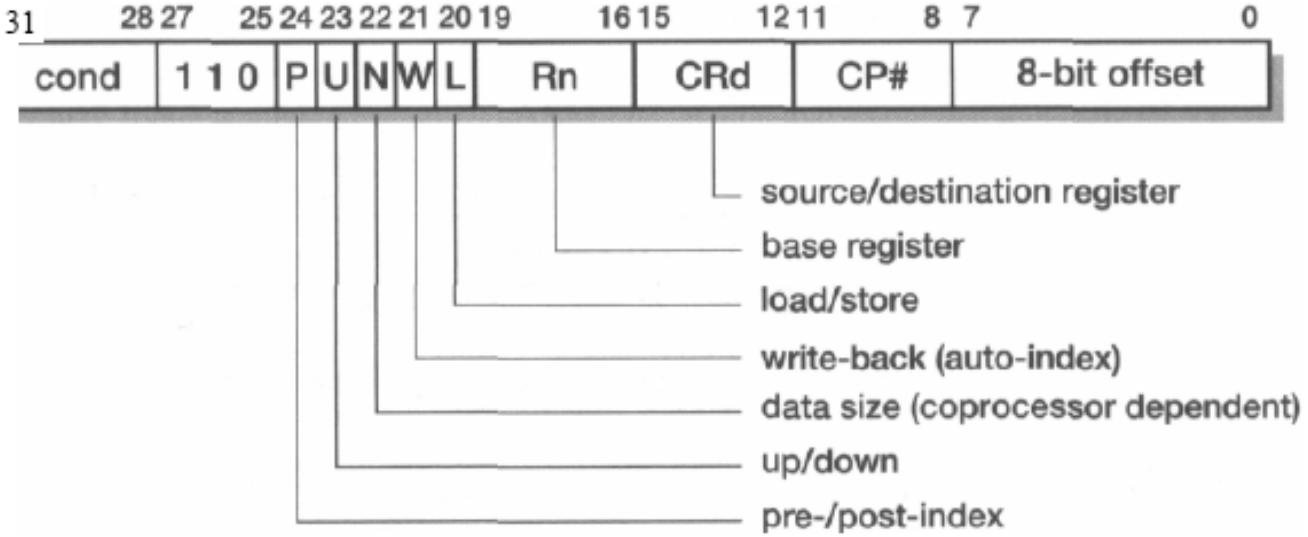



```
CDP{cond} <CP#>, <Cop1>, CRd, CRn, CRm{, <Cop2>}
```

  - La interpretación de la instrucción es dependiente del coprocesador usado

```
CDP       CP15, 2, C12, C10, C3, 4
```

Instrucciones de coprocesador: *Data transfers I*

- Al igual que en la anterior, la instrucción se ofrece a todos los coprocesadores presentes
- → 


```
LDC/STC{cond}{L} <CP#>, CRd, [Rn, <offset>]{!}
LDC/STC{cond}{L} <CP#>, CRd, [Rn], <offset>
```

  - La longitud de la transferencia se establece con el bit “N”, que se activa con el sufijo “L” de la instrucción

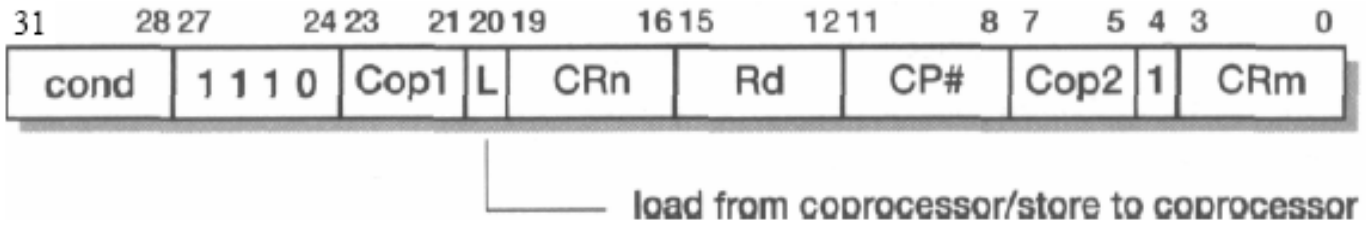
Instrucciones de coprocesador: *Data transfers II*

- Al igual que en la anterior, la instrucción se ofrece a todos los coprocesadores presentes
- La dirección debe estar alineada en palabra
- La utilización del sufijo “L” da la posibilidad al coprocesador de realizar una transferencia múltiple, por ejemplo para cargar datos en doble precisión
- El ARM seguirá generando direcciones consecutivas hasta que el coprocesador termine. Durante este tiempo no se atienden las interrupciones

```
if ConditionPassed(cond) then
    address = start_address
    load Memory[address,4] for Coprocessor[cp_num]
    while (NotFinished(Coprocessor[cp_num]))
        address = address + 4
        load Memory[address,4] for Coprocessor[cp_num]
assert address == end_address
```

Instrucciones de coprocesador: *Coprocessor register transfer I*

- Permite la transferencia de un entero, generado en un procesador, a un registro RPG o CPSR (flags) del ARM
- →




```
MRC{cond} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}
MCR{cond} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}
```

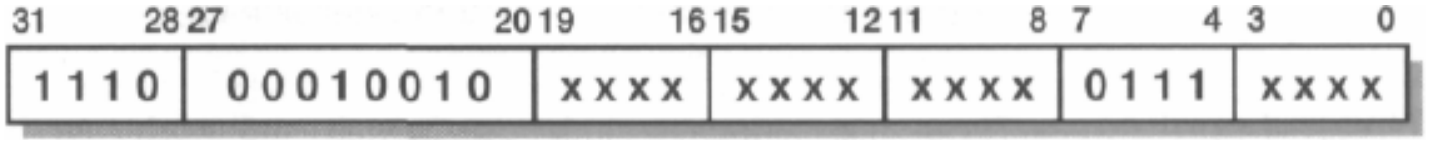
- Si un coprocesador acepta la operación
  - *Load*: El resultado, de 32 bits, de ejecutar Cop1 y Cop2 sobre los operandos CRn y CRm se devuelve sobre el registro Rd
  - *Store*: El coprocesador acepta un dato de 32 bits desde el registro Rd y opera con él
  - Si se especifica PC como Rd, los 4 bits más significativos del dato, se depositan en los flags de condiciones

Instrucciones de coprocesador: *Coprocessor register transfer II*

- Usos:
  - Comparación de números en punto flotante, lo que implica que, como resultado, se devuelven los flags de comparación para actualizar los flags de estado del procesador
  - Operación FLOAT → Envío de un dato desde el ARM y conversión del mismo a punto flotante, depositando este último en un registro del coprocesador
  - Operaciones FIX → conversión de un valor en punto flotante a su correspondiente en punto fijo
    - Esta última operación tiene la característica de necesitar, en determinados casos, ciclos de espera (*busy-wait*) hasta finalizar la conversión
    - Durante los ciclos de espera el ARM puede atender a una petición de interrupción

Instrucción de punto de ruptura

- Utilizada únicamente para la realización de depuración (*debugging*) por software
- →



BKPT

  - Sólo está disponible en procesadores que implementan la arquitectura v5T
  - Su ejecución genera un *Prefetch Abort*
  - Actúa de forma similar al SWI, con un vector de excepción diferente

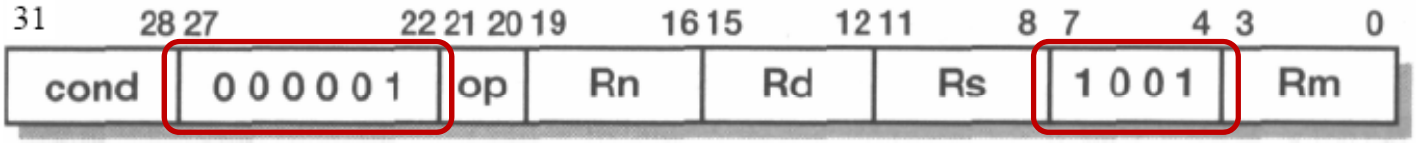
Espacio de instrucciones no utilizadas I

- ARM proporciona un mecanismo para incrementar su juego de instrucciones
  - La extensión del juego de instrucciones puede ocurrir en seis áreas diferentes
    - Instrucciones aritméticas, de control, *load/store*, coprocesador, incondicionales y no definidas
  - En cualquier caso, ante cualquier instrucción no definida, el ARM genera una excepción por instrucción no definida
  - Los bits utilizados para la decodificación son
    - Bits[27:20]; Bits[7:4]
- Espacio de instrucciones no definidas
- → 

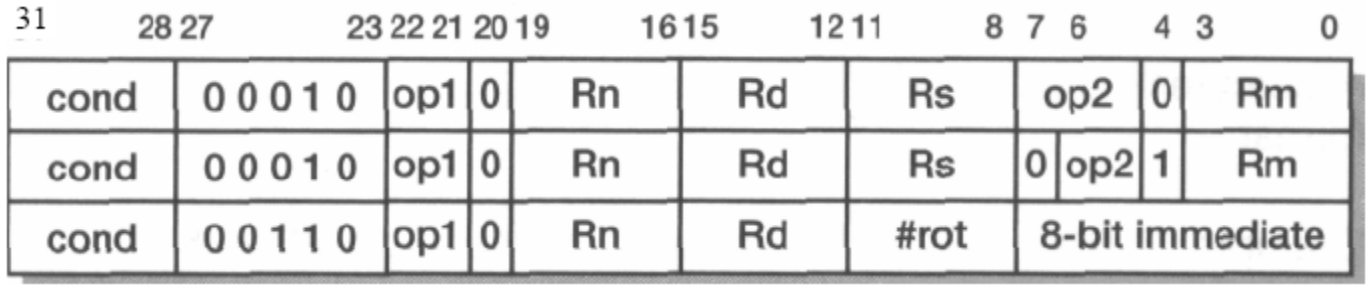
  - Bits[27:25]= 0b011; Bits[4]= 0b1

Espacio de instrucciones no utilizadas II

- Instrucciones aritméticas no usadas
  - →



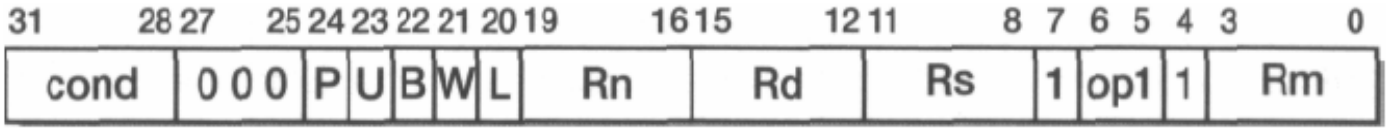
  - Disponen de una codificación similar a las instrucciones de multiplicación
- Instrucciones de control no usadas
  - →



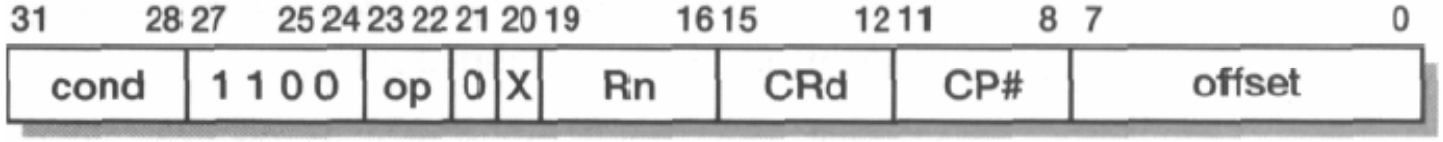
  - Algunas instrucciones que ocupan este espacio son MRS, MSR y BX
  - Espacio ideal para la implementación de instrucciones que afectan al modo de operación del procesador

Espacio de instrucciones no utilizadas III

- Instrucciones *load/store* no usadas
  - →



- Corresponden a este espacio las instrucciones
  - SWAP; LDR/STR *halfword*; LDR/STR *signed byte*
- Instrucciones de coprocesador no utilizadas
  - →



Fallos de memoria I

- Los principales fallos de memoria en el ARM son
  - *Prefetch abort*
    - Ante su aparición, la instrucción se sitúa en el *pipeline* de instrucción
    - Únicamente se actúa cuando ésta entra en la etapa de decodificación. En este momento se genera la correspondiente excepción
    - Si se produce tras un salto, no se provoca ninguna excepción
  - *Data abort*
    - Son más críticas de resolver al tratarse de un error en tiempo de ejecución (no ocurre lo mismo con las anteriores ya que la instrucción no se ha empezado a ejecutar)
    - Si se trata de volver a ejecutar la instrucción, hay que tener en consideración que el estado, al comienzo de la instrucción, debe ser el mismo

Fallos de memoria II

- Existen casos en los que realmente es difícil poder obtener el estado inicial antes de la ejecución
  - *LDM data abort.* Supongamos el caso de la instrucción LDM con registro base r0 y una lista de 16 registros, entre los cuales está el r15
    - El registro base resultará modificado al comienzo de la ejecución
    - Los registros r1 a r14 resultan modificados en la ejecución
    - PC. Su modificación se salva al tener notificación (externa) del fallo de memoria. Al menos tenemos una referencia a la instrucción que produjo el fallo
  - Registro base. Existen dos modelos de implementación del manejo de la excepción por *Data abort* en el ARM
    - *Base restored abort model*. El registro base no se actualiza ante una excepción por *Data abort*
    - *Base updated abort model*. El registro base se actualiza.

Fallos de memoria III

Ejemplo

- El siguiente ejemplo corresponde a un sistema LPC2148 con memoria SDRAM a partir de la dirección 0x7FD0 0000 y un área de memoria reservada dejado de la misma
- Instrucción

```
LDR     r8, 0x7FD0010
STDMDA  r8!, {r0-r7}
```

- Mapa de memoria tras la ejecución
- Valor de r8 tras la ejecución
  - *Base restored*: 0x7FD00000
  - *Base updated*: 0x7FCFFFF0