- MICROCONTROLADORES CORTEX -

ASSEMBLER

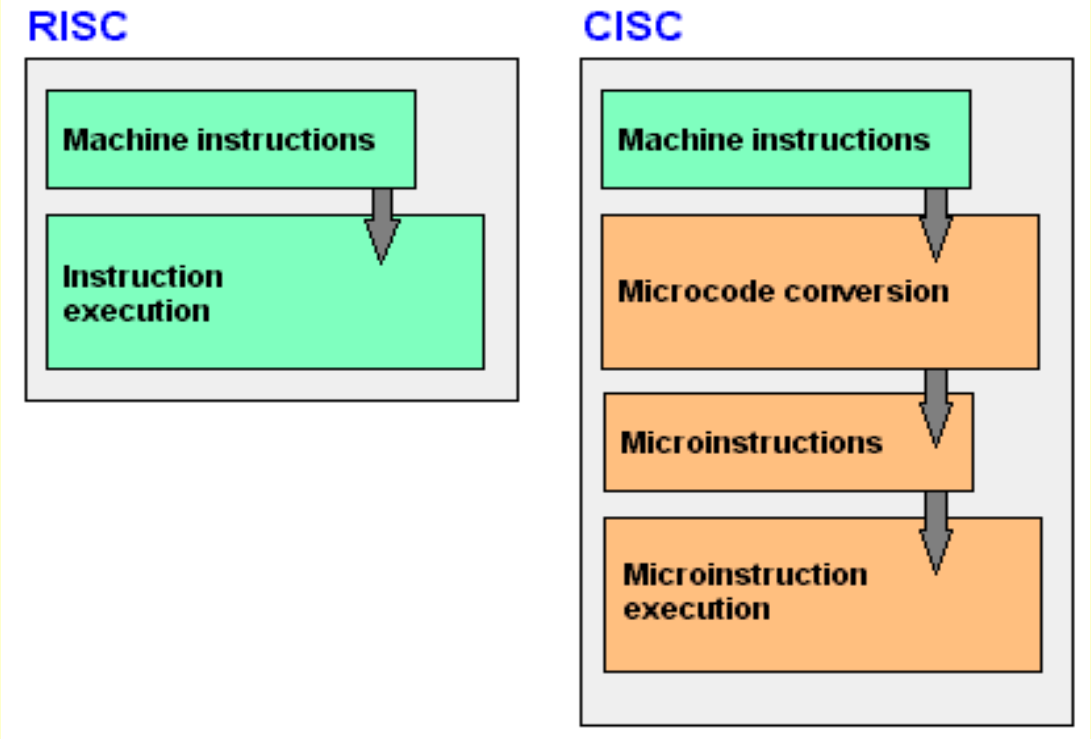
Cátedra: Técnicas Digitales II
UTN – FRBA
Marzo 2019

- MICROCONTROLADORES CORTEX -

CONCEPTOS GENERALES DE INSTRUCCIONES

Clasificación de computadoras

- Complejidad de su repertorio de instrucciones
  - CISC – Complex Instruction Set Computer
  - RISC – Reduced Instruction Set Computer



Análisis estadístico de ejecución de programas estándares

Tipo de instrucción	% de Uso
Movimiento de datos	43
Control de flujo (branches)	23
Operaciones Aritméticas	15
Comparaciones	13
Operaciones Lógicas	5
Otras	1

Ventajas de Arquitectura RISC

1. Mejor aprovechamiento de área de silicio.
2. Menor consumo de energía.
3. Menor tiempo de desarrollo
4. Mayor rendimiento. Típicamente 40 a 50% menor consumo por MHz de reloj.

Inconvenientes de Arquitectura RISC

1. Generalmente tienen una pobre densidad de código comparada con CISC
2. Se requerirán múltiples instrucciones RISC para ejecutar una CISC (aunque el tiempo de ejecución del programa completo en RISC sea menor que el tiempo de ejecución en CISC)

El repertorio de instrucciones con que nos encontraremos será:

1. Instrucciones de procesamiento de datos
2. Instrucciones de movimientos de datos
3. Instrucciones de control de flujo
4. Instrucciones especiales
5. Otras

Cortex: Implementación de RISC

Optimización de las:

- Instrucciones de Procesamiento de datos
- Instrucciones de Transferencia de Datos
- Instrucciones de Control de Flujo

Características RISC adoptadas por Familia Cortex

1. Tamaño fijo de instrucciones de 32 bits.
2. Optimización de la ejecución de las instrucciones más frecuentes.
3. Arquitectura de almacenamiento y carga (*load-store*).
4. Importante banco de registros.
5. Estructura pipeline (y posteriormente superescalar) en lugar del microcódigo de CISC.

Principios Básicos de Familia Cortex

- Se basa en Arquitectura RISC.
- Arquitectura load-store.
- Pipeline
- Instrucciones de tamaño fijo: 32 bits
- Ejecución condicional de todas las instrucciones, para maximizar el rendimiento de la ejecución.
- Computadora de 3 direcciones.
- Varios modos de operación.
- Las operaciones aritméticas y lógicas **son entre registros.**

Principales características de Familia Cortex

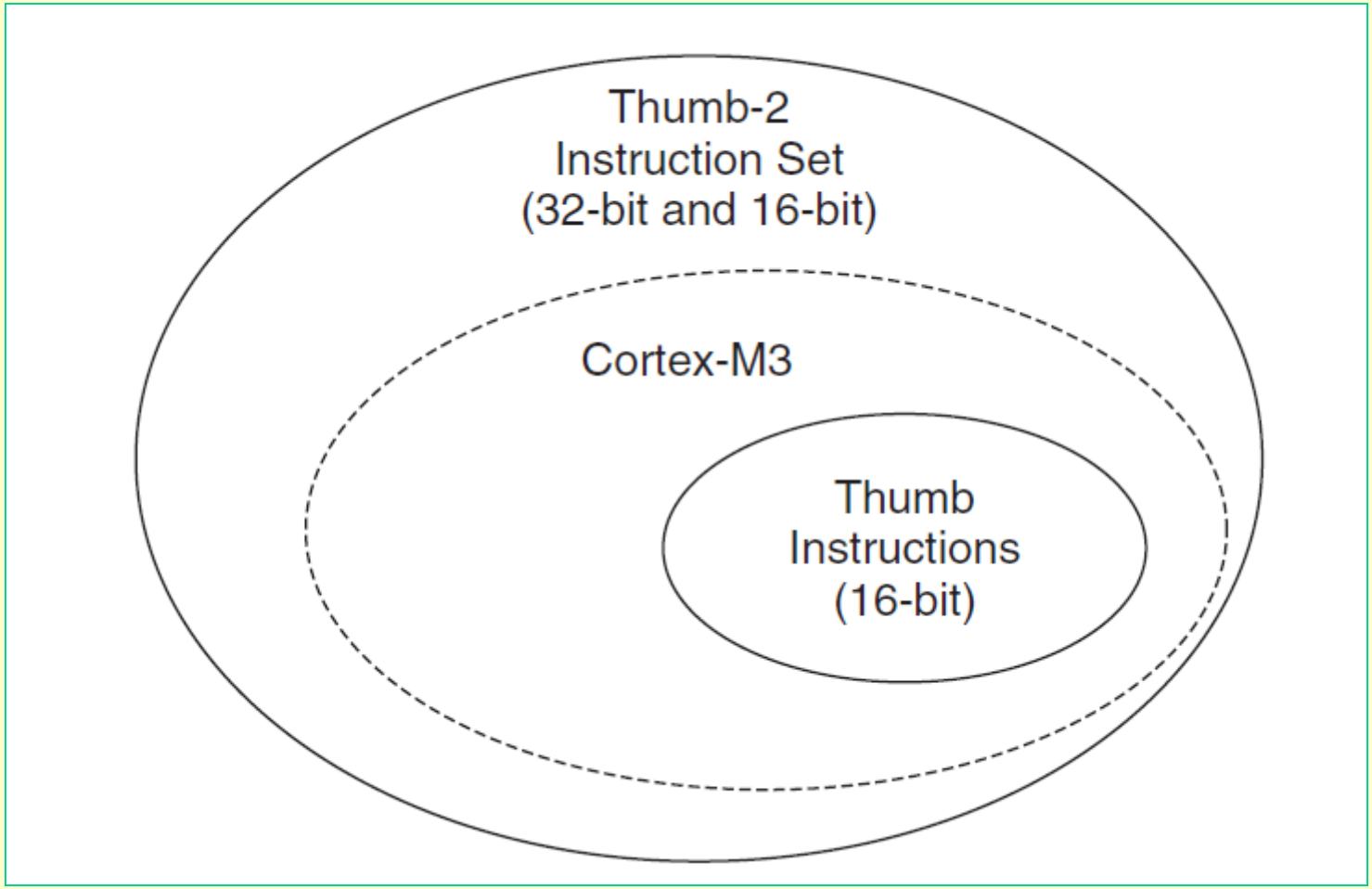
- Arquitectura RISC
- Minimizar la cantidad de transistores
- Minimizar el consumo

Tamaño de instrucciones y datos

- Cortex es una arquitectura de 32-bits.
  - **Byte** → 8 bits
  - **Halfword** → 16 bits
  - **Word** → 32 bits

- El repertorio de modos de instrucciones
  - 32-bit **ARM**
  - 16-bit **Thumb**
  - **Thumb-2** es una combinación de ambos

Thumb-2 y Thumb



Instrucciones en Thumb-2

- En Unified Assembler Language muchas instrucciones se representan nemónicamente de igual manera para 32 ó 16 bits.
- Si no indicamos lo contrario, y de ser posible los compiladores habitualmente eligen la opción de 16 bits.

  Ej: ADDS R0, #1 será implementada en 16 bits
- Si deseamos ser más explícitos (Strong Typing)

```
ADDS.W  R0,#1     ; Wide = 32 bits
ADDS.N  R0,#1     ; Narrow = 16 bits
```

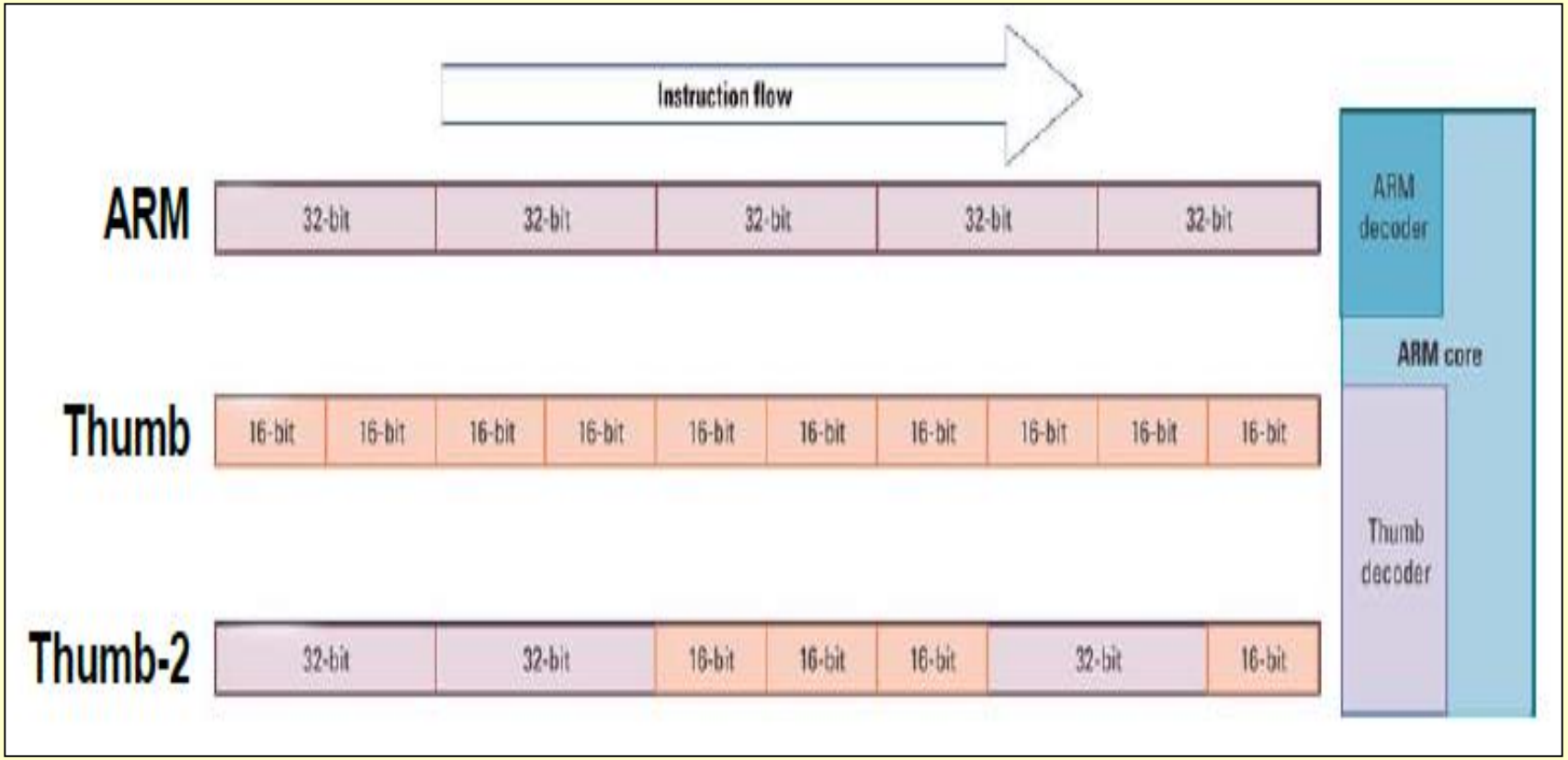
Thumb-2

- Permite mezclar instrucciones de 16 bits de longitud con instrucciones de 32 bits.
- La mayor parte de las instrucciones de los Cortex son de 16 bits.
- Internamente se procesan como instrucciones de 32 bits, por lo que existe una etapa de decompresión de las instrucciones que las lleva de 16 a 32 bits.

Thumb-2

- Instrucciones de Longitud Variable.
  - Instrucciones ARM son de 32 bits
  - Instrucciones Thumb son de 16bits
  - Instrucciones Thumb-2 pueden ser indistintamente de 16 ó 32 bits
- Thumb-2 brinda un 26% de mejora de densidad de código que ARM
- Thumb-2 un 25% de mejora de densidad de código que Thumb

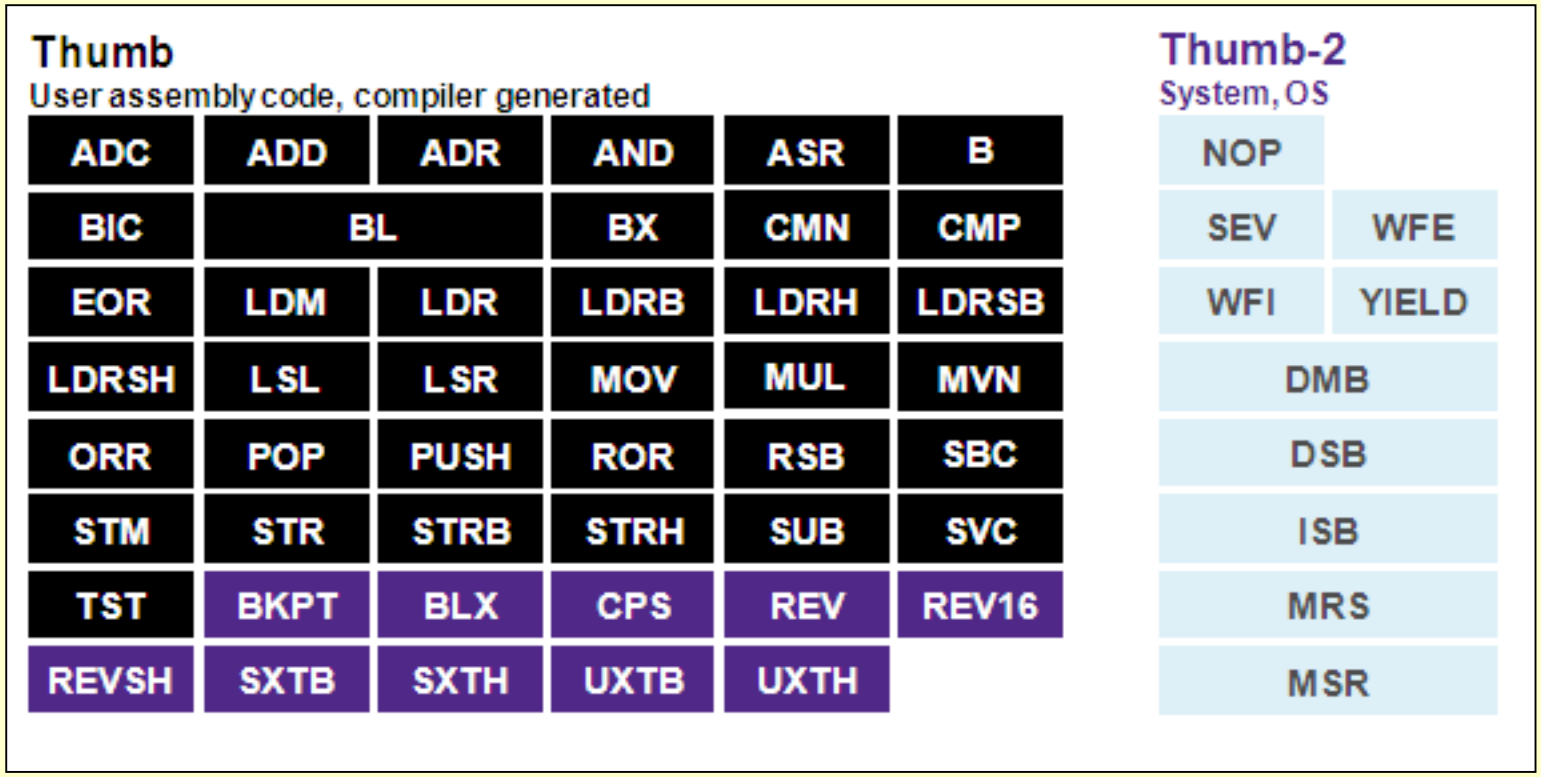
Thumb-2


Instruction flow
ARM
32-bit
32-bit
32-bit
32-bit
32-bit
Thumb
16-bit
16-bit
16-bit
16-bit
16-bit
16-bit
16-bit
16-bit
16-bit
16-bit
Thumb-2
32-bit
32-bit
16-bit
16-bit
16-bit
32-bit
16-bit
ARM decoder
ARM core
Thumb decoder

- MICROCONTROLADORES CORTEX -

REPERTORIO DE INSTRUCCIONES

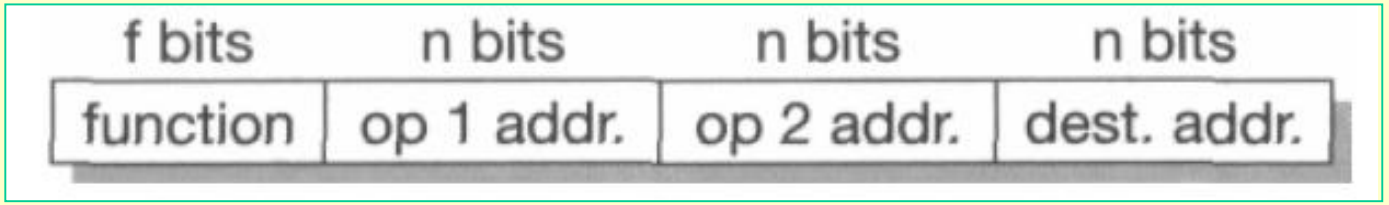
Modos de Instrucciones



Instrucciones

- **Formatos de instrucción de 3 direcciones**
- Consta de “f” bits para el código de operación, “n” bits para especificar la dirección del 1er. operando, “n” bits para especificar la dirección del 2do. operando y “n” bits para especificar la dirección del resultado (el destino). El formato de esta instrucción en *Assembler*, por ejemplo para la instrucción de sumar dos números para producir un resultado, es:

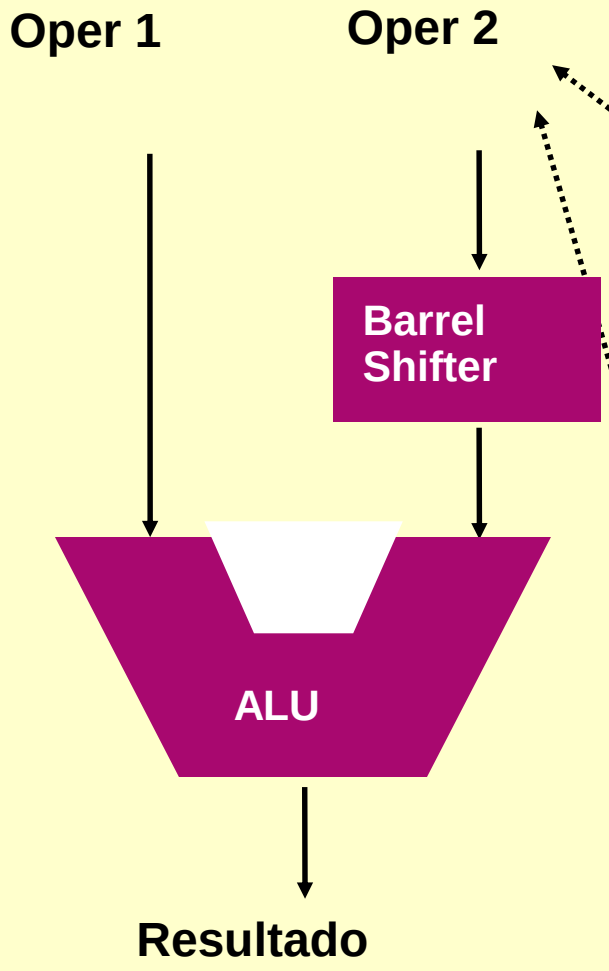
```
ADD   d, s1, s2          ;d := s1 + s2.
```



Instrucciones

- **Casi todas las instrucciones se ejecutan en un ciclo de reloj**
- **Modos de direccionamiento simples**
- El procesamiento de datos solo opera con contenidos de registros, no directamente en memoria.
- **Control sobre la Unidad Aritmética Lógica (ALU, *Arithmetic Logic Unit*) y el "*Shifter*",** en cada instrucción de procesamiento de datos para maximizar el uso de la ALU y del "*shifter*".
- **Modos de direccionamiento con incremento y decremento automático de punteros,** para optimizar los lazos de los programas.
- **Carga y almacenamiento de múltiples registros,** para maximizar el rendimiento de los datos.

Barrel Shifter: El 2º Operando



- Registro, opcionalmente con operación de desplazamiento
  - El valor del desplazamiento puede ser:
    - 5 bit entero no signado
    - Especificado en el LSB de otro registro.
  - Usado para multiplicar por una constante
- Valor inmediato
  - 8 bit : Número en el rango de 0-255.
    - Rotado a la derecha en un nùmero par de posiciones
  - Permite que sean cargadas constantes directamente a los registros de 32-bits

Instrucciones de procesamiento de datos

Instrucciones de procesamiento de datos

Instruction	Function
ADC	Add with carry
ADD	Add
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear (Logical AND one value with the logic inversion of another value)
CMN	Compare negative (compare one data with two's complement of another data and update flags)
CMP	Compare (compare two data and update flags)
CPY	Copy (available from architecture v6; move a value from one high or low register to another high or low register)
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MOV	Move (can be used for register-to-register transfers or loading immediate data)

Instrucciones de procesamiento de datos

MUL	Multiply
MVN	Move NOT (obtain logical inverted value)
NEG	Negate (obtain two's complement value)
ORR	Logical OR
ROR	Rotate right
SBC	Subtract with carry
SUB	Subtract
TST	Test (use as logical AND; Z flag is updated but AND result is not stored)
REV	Reverse the byte order in a 32-bit register (available from architecture v6)
REVH	Reverse the byte order in each 16-bit half word of a 32-bit register (available from architecture v6)
REVSH	Reverse the byte order in the lower 16-bit half word of a 32-bit register and sign extends the result to 32 bits. (available from architecture v6)
SXTB	Signed extend byte (available from architecture v6)
SXTH	Signed extend half word (available from architecture v6)
UXTB	Unsigned extend byte (available from architecture v6)
UXTH	Unsigned extend half word (available from architecture v6)

Procesamiento de Datos

- Consiste de:
  - Aritmeticas: **ADD ADC SUB SBC RSB RSC**
  - Logicas: **AND ORR EOR BIC**
  - Comparaciones: **CMP CMN TST TEQ**
  - Movimiento Datos: **MOV MVN**

- Estas instrucciones operan sobre registros y NO en memoria.

- Sintaxis:

  <Operación>{<cond>}{S} Rd, Rn, Operand2

  - Comparaciones sólo afectan flags – no especifican Rd
  - Movimiento de datos no especifican Rn

- El segundo operando se envía a la ALU a través del desplazador en barril (shifter).

Procesamiento de Datos

Aritméticas

- ADD r1, r2, r3 ; r1 = r2 + r3
- ADC r1, r2, r3 ; r1 = r2 + r3 + C
- SUB r1, r2, r3 ; r1 = r2 - r3
- SUBC r1, r2, r3 ; r1 = r2 - r3 + C - 1
- RSB r1, r2, r3 ; r1 = r3 - r2 (inversa)
- RSC r1, r2, r3 ; r1 = r3 - r2 + C - 1

Suma Multibyte

- Se pretende sumar dos magnitudes de 64 bits cada una de ellas están en registros según el siguiente esquema:

r1.r2.r3 = r4.r5 + r6.r7

- Donde r1 contendrá el acarreo de las sumas anteriores

Aritméticas – Suma Multibyte

```
Inicio
; Inicialización de Registros
        mov     r4,#0x50
        mov     r5,#0x70
        mov     r6,#0xc0
        mov     r7,#0xf0
        mov     r1,#0               ;Acarreo inicial = 0
; Suma
        adds    r3,r5,r7
        adcs    r2,r4,r6
        bcc     no_hubo_acarreo
        add     r1,r1,#0x1
no_hubo_acarreo
```

Procesamiento de Datos

- Aritméticas
  - ADD r3,r2,#1
  - ADD r3,r2,r1, lsl #3 (lsr, asl, asr, ror, rrx)
  - ADD r5,r5,r3, LSL r2
  - MUL r4,r3,r2
  - MLA r4,r3,r2,r1 ;r4:=(r3 x r2 + r1)
  - RSB r0,r0,r0, LSL #3
    - » ; Multiplicar por 7

Multiplicación

Calculation	Result	Clock Cycles
16b x 16b	32b	1
32b x 16b	32b	1
32b x 32b	32b	1
32b x 32b	64b	3-7*

Procesamiento de Datos

- Lógicas
  - AND r0,r1,r2 ; r0:= r1.r2
  - ORR r0,r1,r2 ; r0:= r1 + r2
  - EOR r0,r1,r2 ; r0:= r1 xor r2
  - BIC r0,r1,r2 ; r0:= r1 and not r2
  - AND r8,r7,#0xff ; r8:= r7 . 0x000000ff

Procesamiento de Datos

Solo afectan los Flags

```
CMP     r1, r2      ; cc por r1 - r2
CMN     r1, r2      ; cc por r1 + r2
TST     r1, r2      ; cc por r1 and r2
TEQ     r1, r2      ; cc por r1 xor r2
```

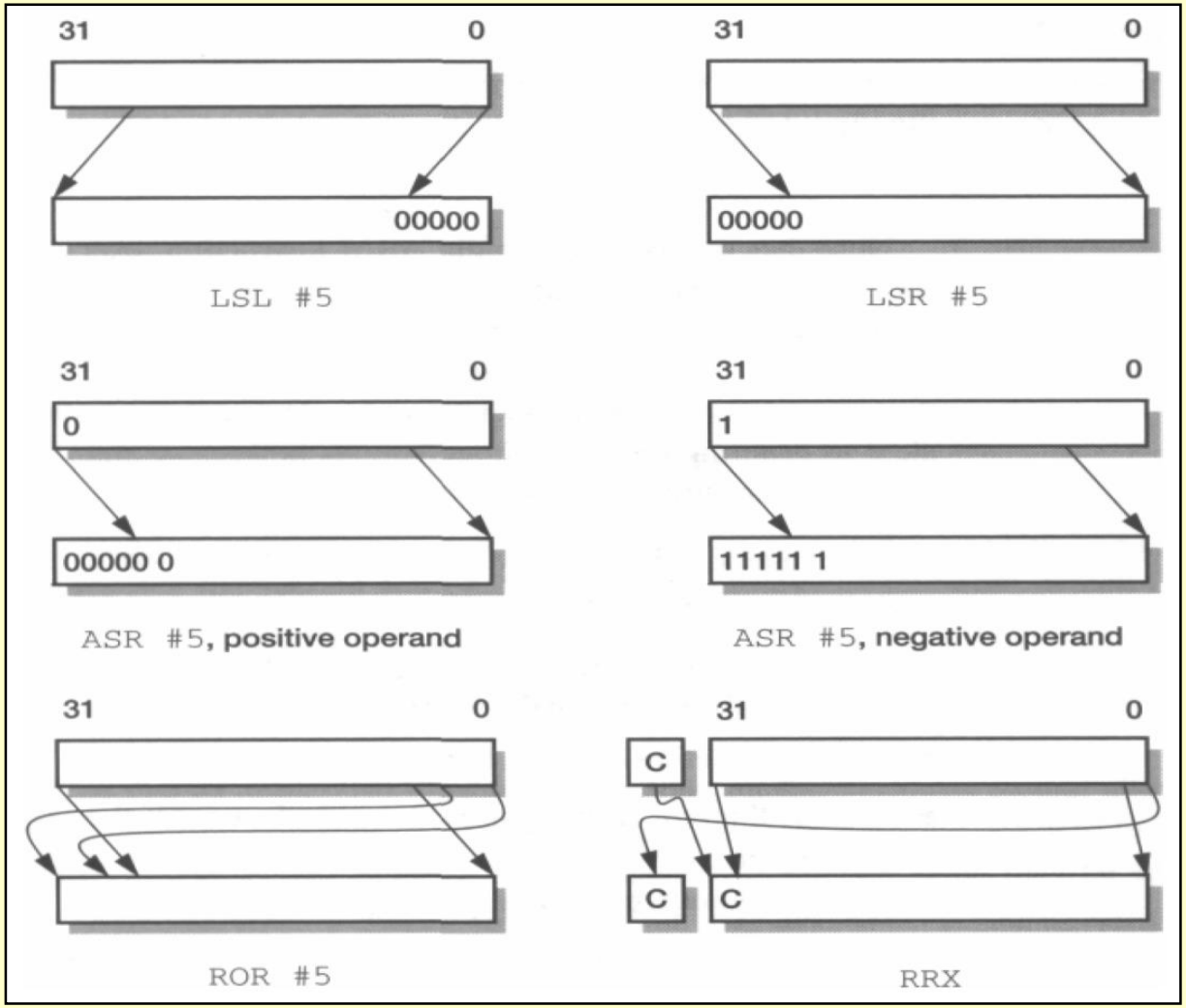
Procesamiento de Datos
Inmediatas

- ADD r3, r3,#1 ; r3 := r3 + 1
- AND r8, r7,#oxff ; r8 := r7[7:0]

Procesamiento de Datos
Desplazamientos

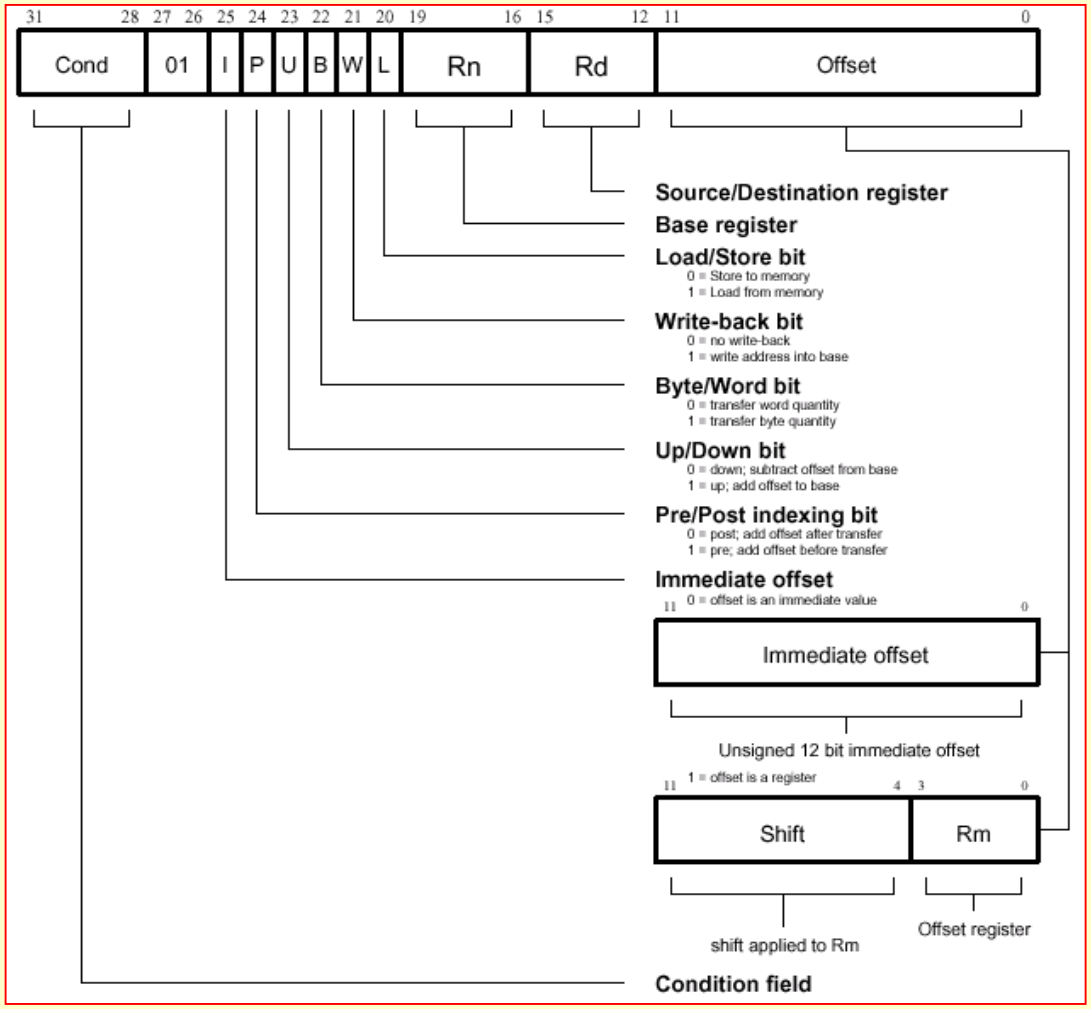
- ADD r3, r2, r1, LSL #3 ; r3 := r2 + 8 x r1
- ADD r5, r5, r3, LSL r2 ; r5 : = r5 + r3 x 2^{r2}
- RSB r0,r0,r0, LSL #3
    - ; Multiplicar por 7

Desplazamientos



Instrucciones de transferencia de datos

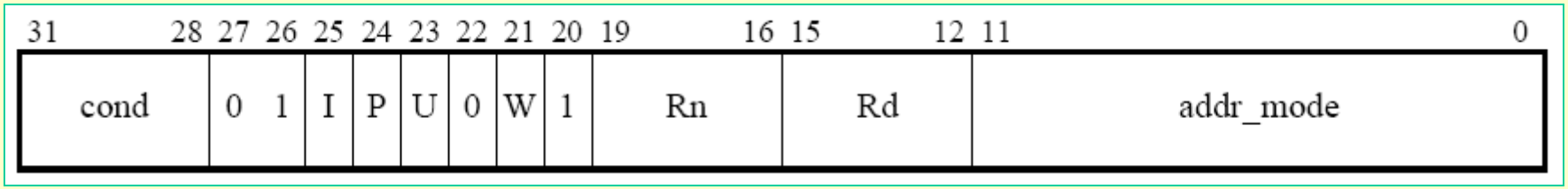
Transferencias de Datos



Transferencias de Datos

- Movimiento
  - MOV r0,r2
  - MVN r0,r2 ; r0:= not r2

Transferencias de Datos
LDR




```
LDR     r0,[r1]              ; r0 := mem32[r1]
LDR     R4, [R2, #4]         ; Carga word en R4 desde direc R2 + 4
LDR     R4, [R2, R1]         ; Carga word en R4 desde direc R2 + R1
LDRH R3, [R6, R5]            ; Carga half word en R3 desde R6 + R5
LDRB    R2, [R1, #5]         ; Carga byte en R2 desde R1 + 5
LDR     R6, [PC, #0x3FC] ; Carga R6 desde PC + 0x3FC
LDR     R5, [SP, #64]        ; Carga R5 desde SP + 64
```

Transferencias de Datos

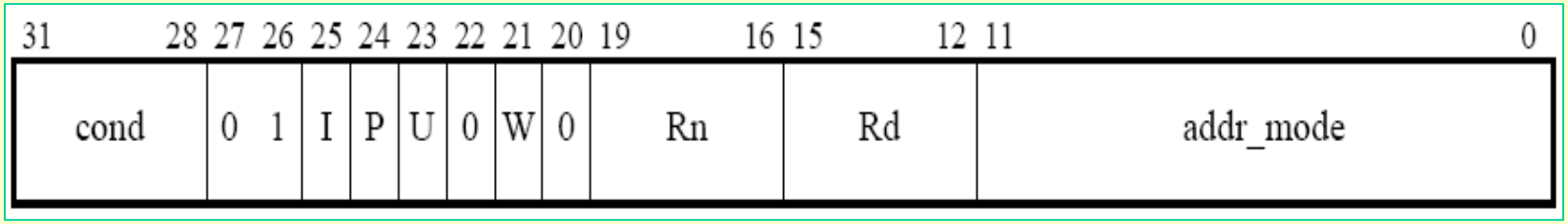
Pre y post indexado

```
LDR r0,[r1,#4]     ; r0 := men32[r1+ 4]

LDR r0,[r1,#4]!    ; r0 := mem32[r1+ 4] ; r1 := r1 + 4
```

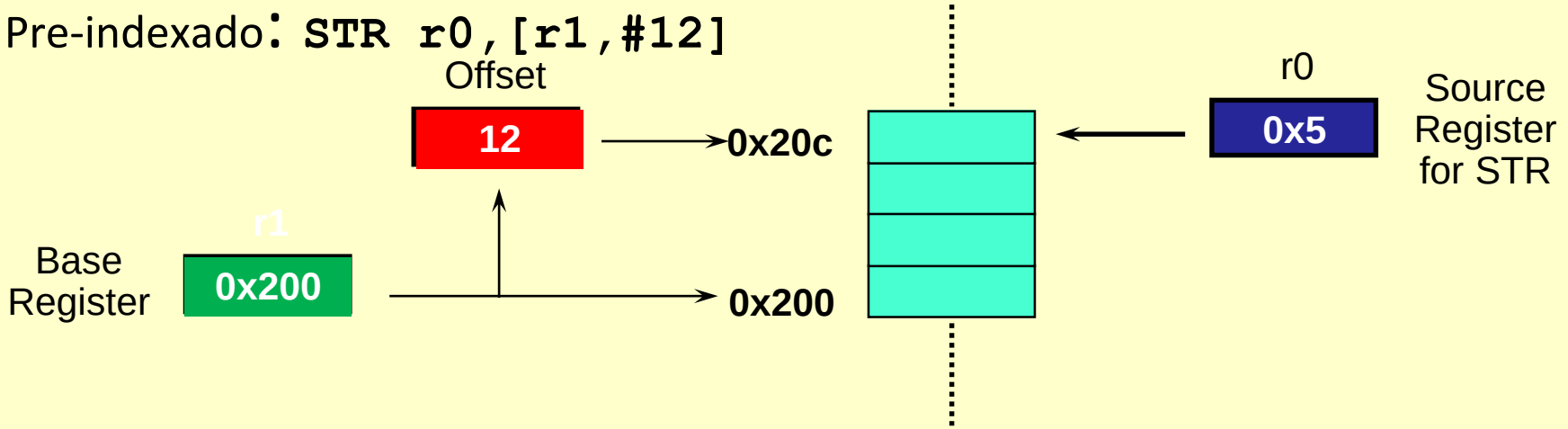
Transferencias de Datos

STR

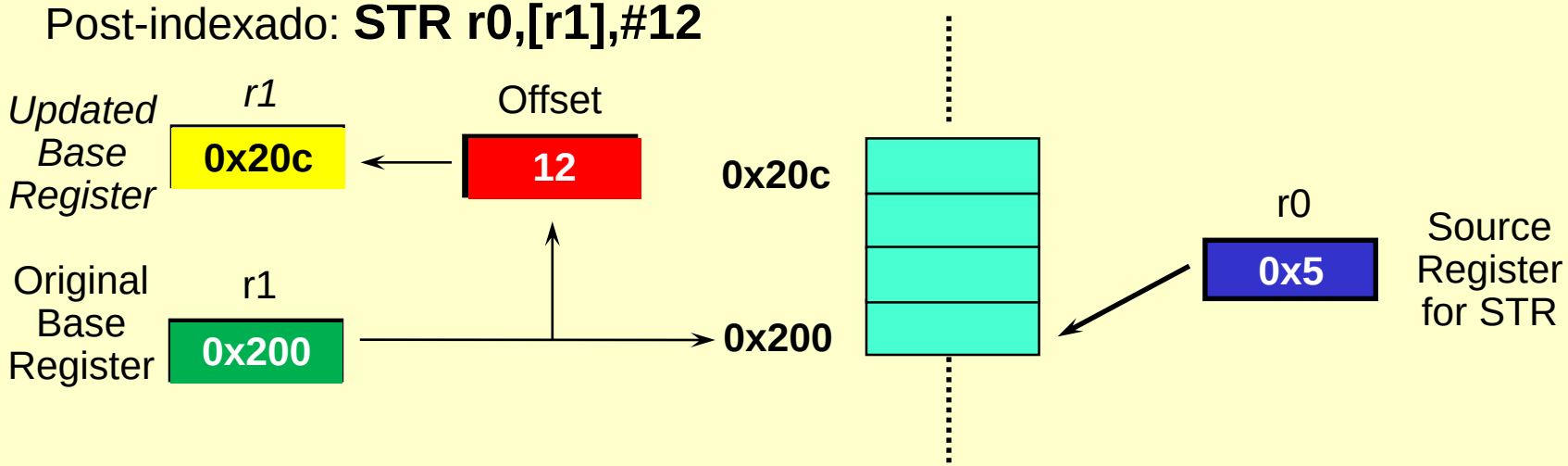



```
STR     r0,[r1]              ; mem32[r1] := r0
STR     R0, [R7, #0x7C]      ; Guarda word desde R0 a direc. R7 + 124
STRB    R1, [R5, #31]        ; Guarda byte desde R1 a direc. R5 + 31
STRH    R4, [R2, R3]         ; Guarda halfword desde R4 a dir R2 + R3
STR     R4, [SP, #0x260]     ; Guarda R4 a direc. SP + 0x260
```

Preindexado y postindexado



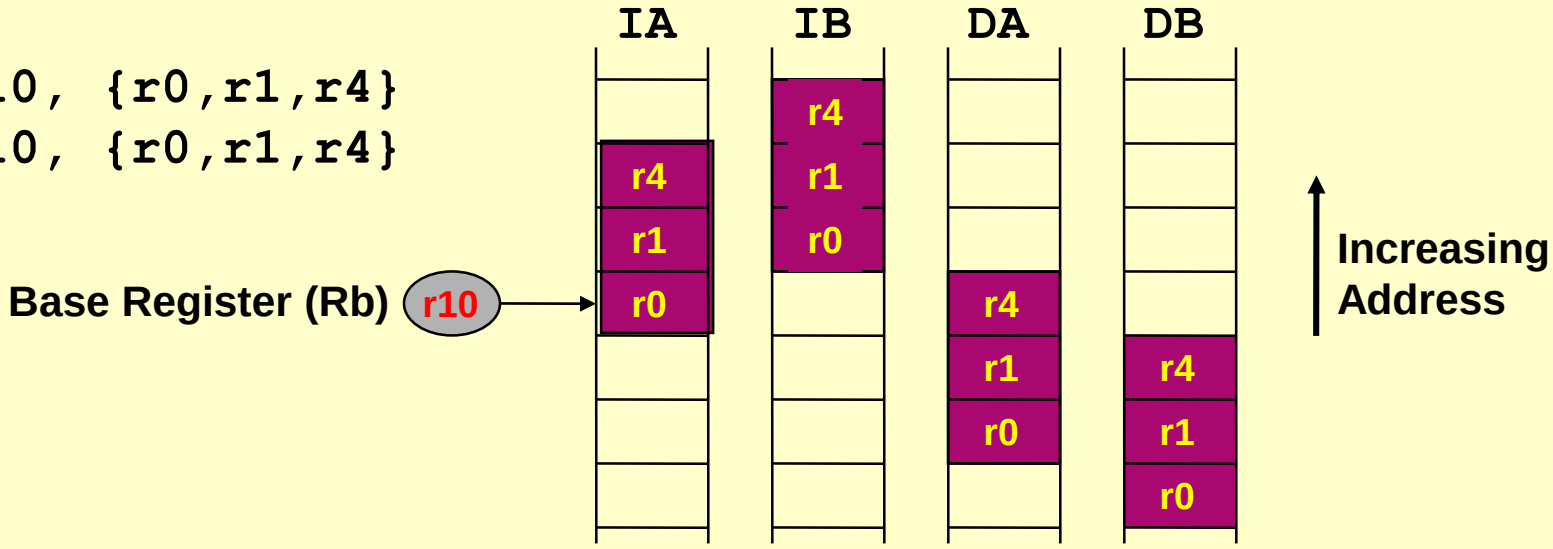
- Post-indexado: **STR r0,[r1],#12**



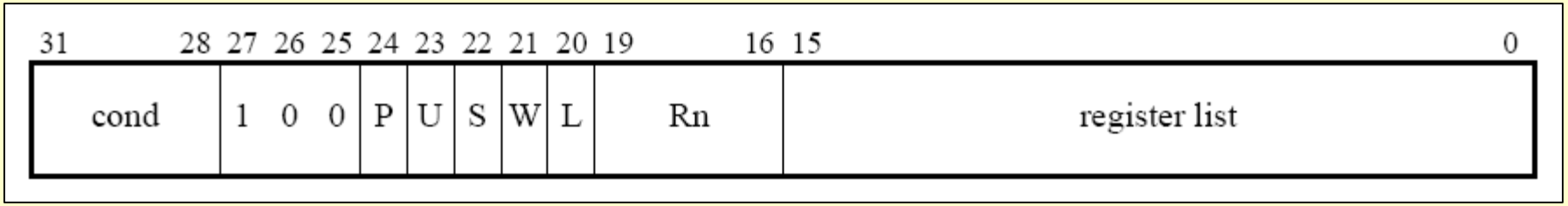
Multiples Load y Store Registros

- Sintaxis:
  - **<LDM|STM>**{<cond>}<addressing_mode> Rb{!}, <register list>
- 4 modos de direccionamiento:
  - **LDMIA / STMIA** increment after
  - **LDMIB / STMIB** increment before
  - **LDMDA / STMDA** decrement after
  - **LDMDB / STMDB** decrement before

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



LDMIA y STMIA



LDMIA R7!, {R0-R3, R5} ; Load R0 to R3-R5 desde R7, add 20 to R7
STMIA R0!, {R3, R4, R5} ; Store R3-R5 to R0: add 12 to R0

PUSH y POP

Función:

```
PUSH {R0-R7, LR}  ; push al stack (R13) R0-R7 y la
                  ; dirección de retorno
... ; Cuerpo de la función
...
POP {R0-R7, PC} ; restaura R0-R7 del stack
                ; y el PC y retorna
```

Uso de la Pila

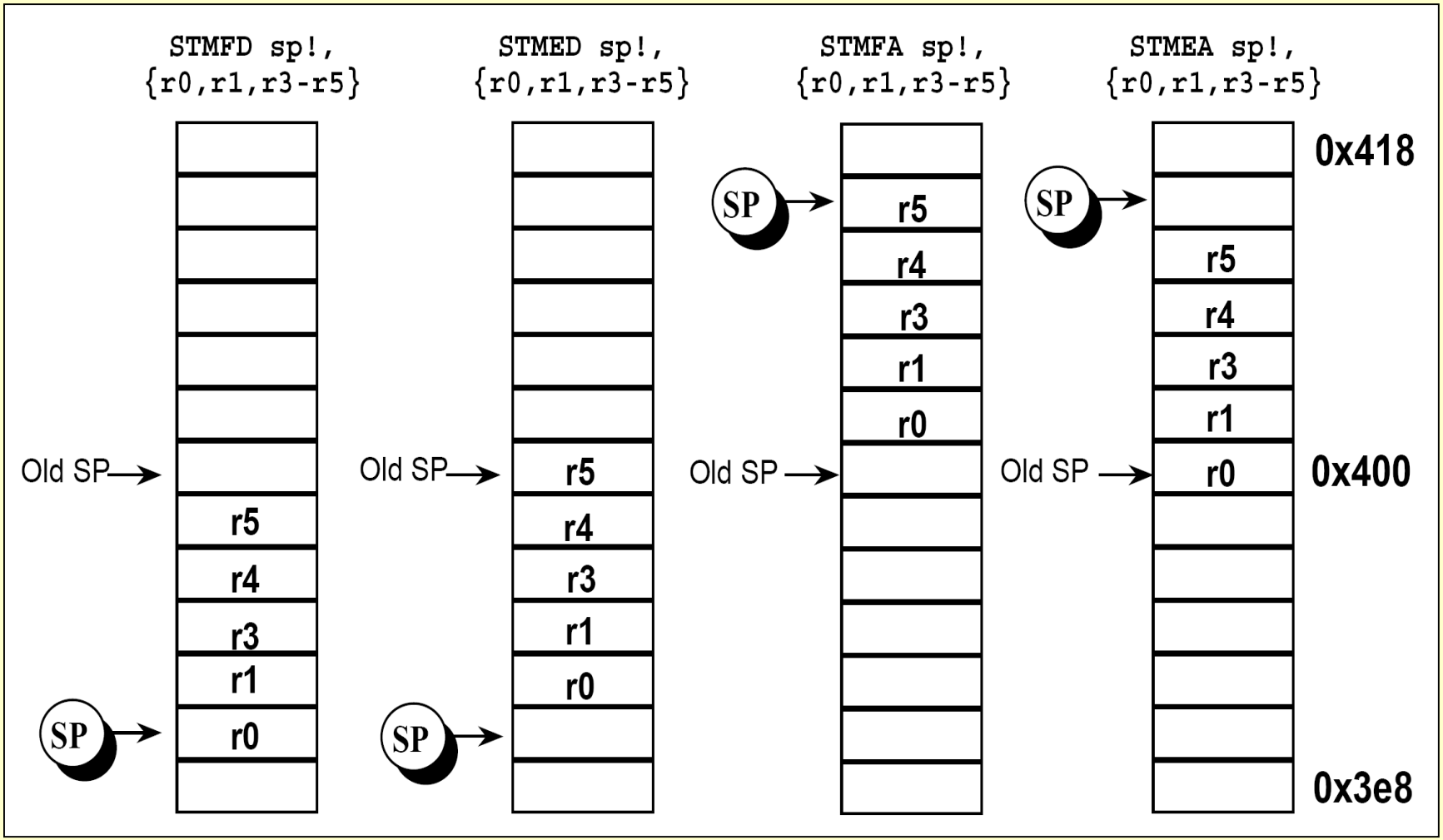
```
LDMIA  r0!, {r2-r9}
STMIA  r1,  {r2-r9}
```

```
STMFD   r13!,   {r2-r9}
LDMIA   r0!,    {r2-r9}
STMIA   r1,     {r2-r9}
LDMFD   r13!,   {r2-r9}
```

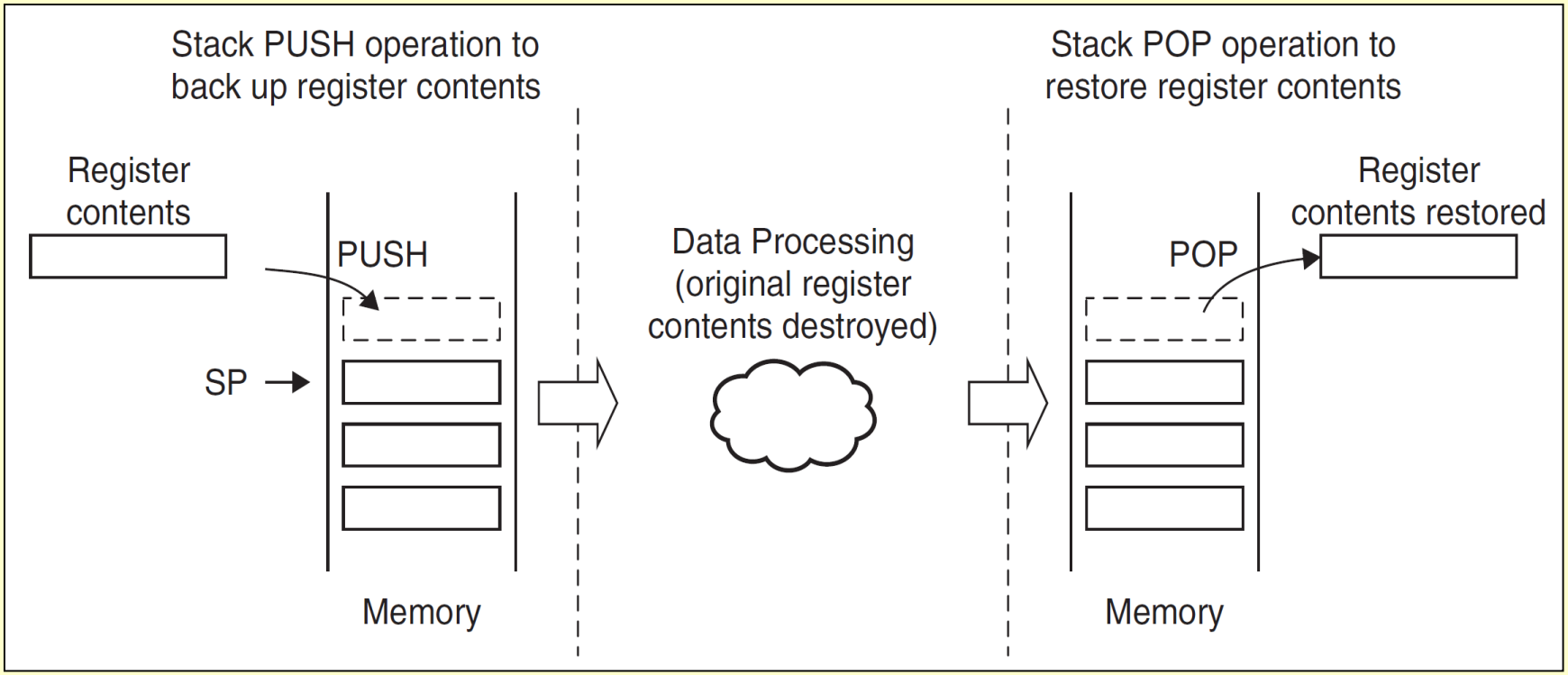
Uso de la Pila

- STMFD / LDMFD : Full Descending stack
- STMFA / LDMFA : Full Ascending stack.
- STMED / LDMED : Empty Descending stack
- STMEA / LDMEA : Empty Ascending stack

Uso de la Pila

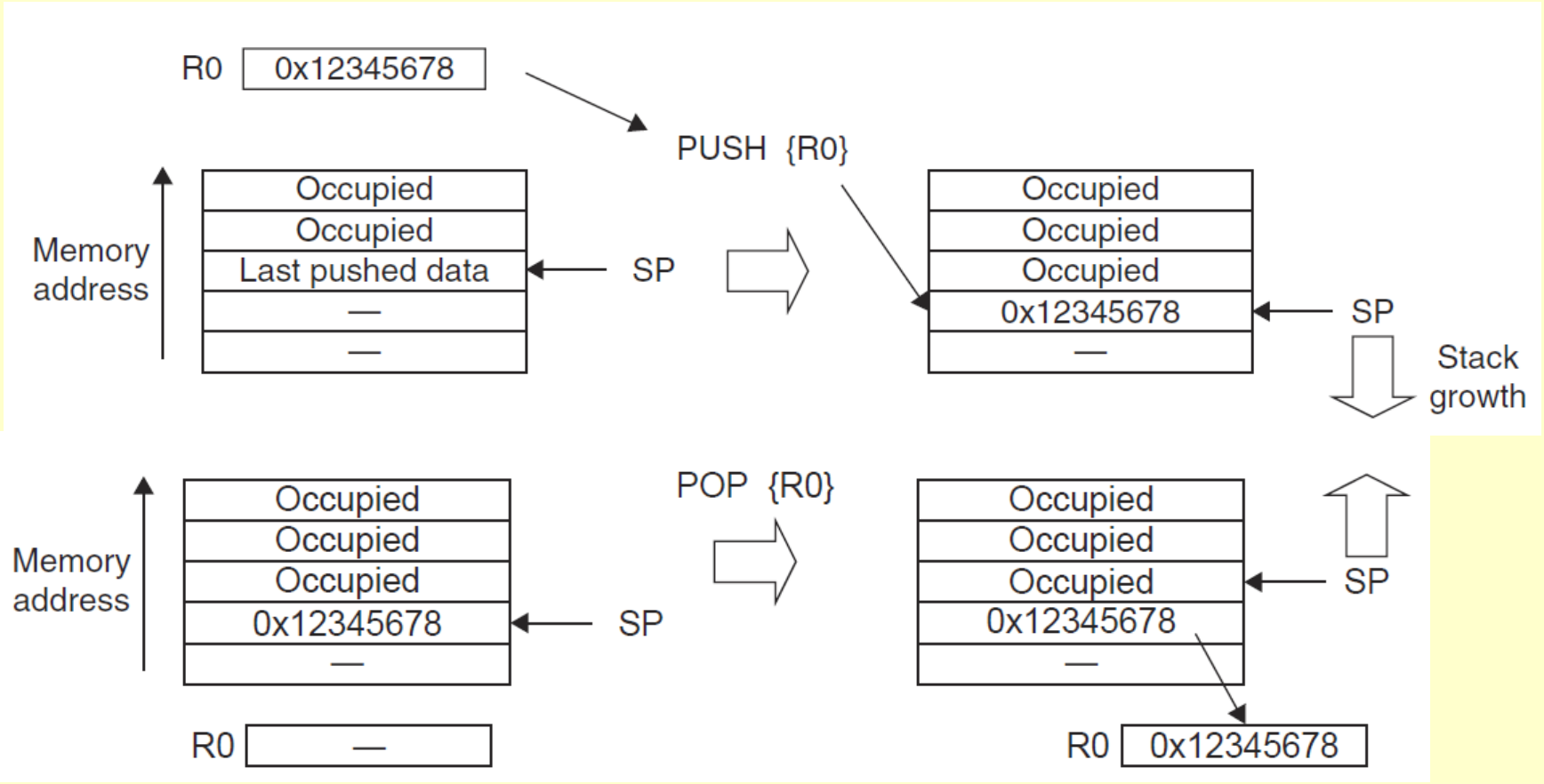


PUSH y POP




```
PUSH  {R0}        ; R13 = R13 - 4, luego Memory[R13] <- R0
POP   {R0}        ; R0  <- Memory[R13], luego R13  = R13 + 4
```

PUSH y POP Ampliados



Uso de la pila en subrutinas

```
; R0 = X, R1 = Y, R2 = Z
BL funcion1
; Regresa al programa ppal
; R0 = X, R1 = Y, R2 = Z
... ;
```

```
funcion1
    PUSH {R0} ; almacena R0 en pila y ajusta SP
    PUSH {R1} ; almacena R1 en pila y ajusta SP
    PUSH {R2} ; almacena R2 en pila y ajusta SP
... ; Ejecuta tarea. (R0, R1 y R2 pueden cambiar)
    POP {R2}  ; restaura R2 y reajusta SP
    POP {R1}  ; restaura R1 y reajusta SP
    POP {R0}  ; restaura R0 y reajusta SP
    BX LR     ; retorna al programa ppal
```

-Program Status Register – Ampliado y Condensado

	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						

- APSR - Application Program Status Register – ALU flags
- IPSR - Interrupt Program Status Register – Interrupt/Exception No.
- EPSR - Execution Program Status Register
  - IT field – If/Then block information
  - ICI field – Interruptible-Continuable Instruction information

	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				

Condensado Almacenado en el stack al Inicio de excepción

Instrucciones de control de flujo

Instrucciones de control de flujo

Instruction	Function
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR
BLX	Branch with link and change state (BLX <reg> only)[1]
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

Saltos - Branch

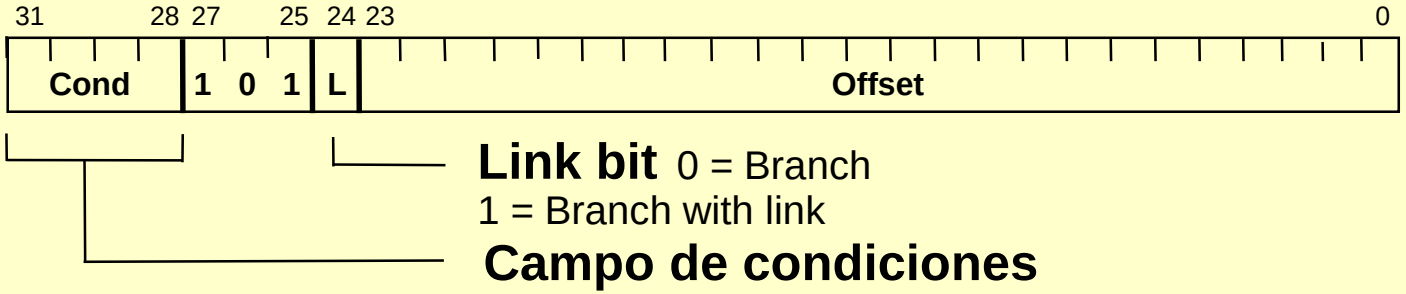
- Instrucciones de salto (Branching): BX, B, BL
- B: salto con desplazamiento de 24 bits con signo
- BL: enlace (link) PC -> R14
- Instrucciones de transferencia de datos: LDR, STR, LDRH, STRH, LDRSB, LDRSH, LDM, STM, SWP.

Control de flujo

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

Instrucciones de Branch

- Branch : `B{<cond>} etiqueta`
- Branch with Link : `BL{<cond>} Etiqueta subrutina`

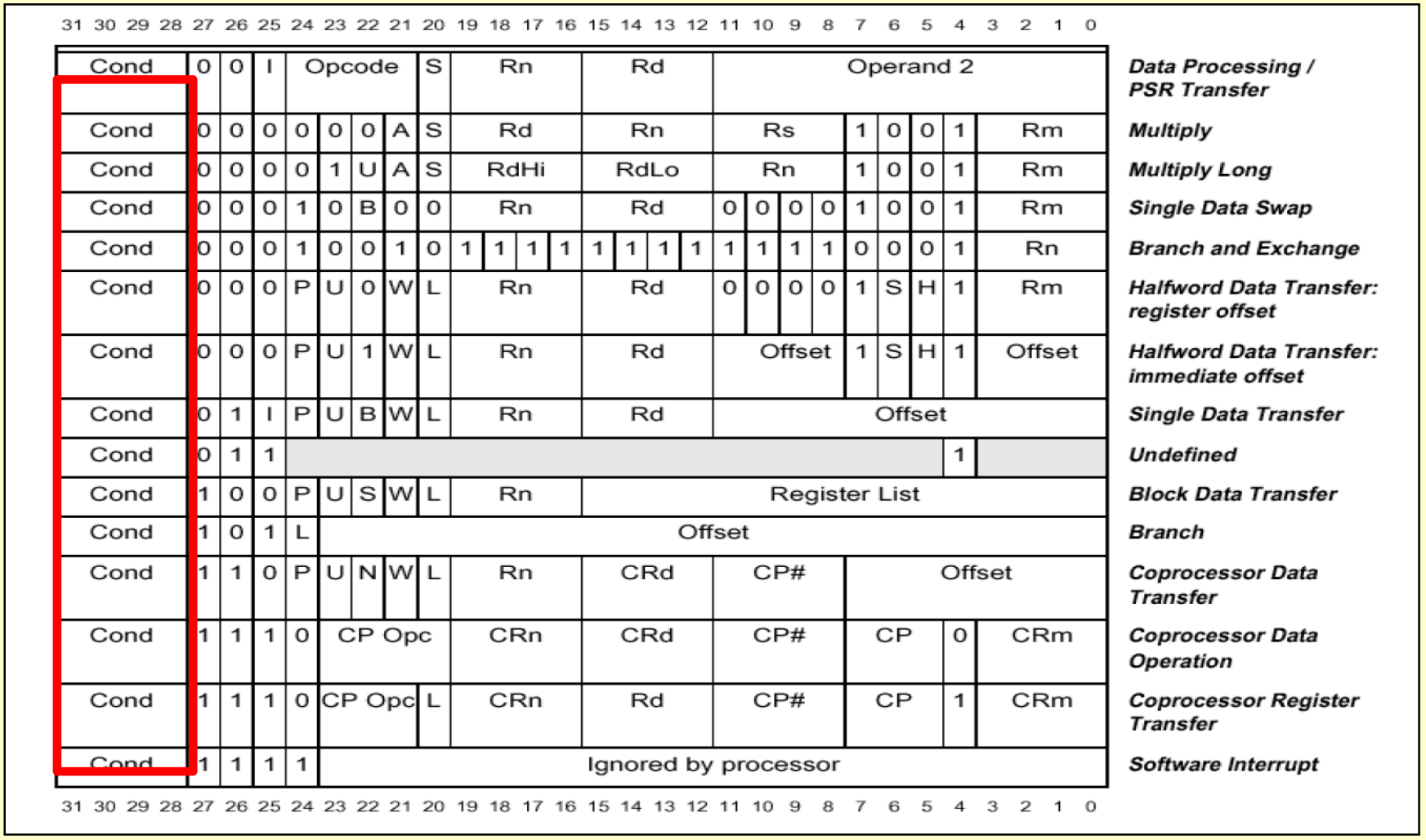


- El procesador desplaza el campo del offset a la izquierda en dos lugares, extiende el signo y lo suma al PC
  - ± 32 Mbyte de rango
  - ¿Cómo se pueden implementar Branches mayores?

Subrutinas

```
   BL       Subru
…
Subru: ….
   MOV      PC,R14
```

Instrucciones Condicionales



Ejemplos condicionales

Fuente C

```
if (r0 == 0)
{
  r1 = r1 + 1;
}
else
{
  r2 = r2 + 1;
}
```

Instruciones

Incondicional

```
...
CMP r0, #0
  BNE else
  ADD r1, r1, #1
  B end
else
  ADD r2, r2, #1
end
```

- 5 instrucciones
- 5 palabras
- 5 o 6 ciclos

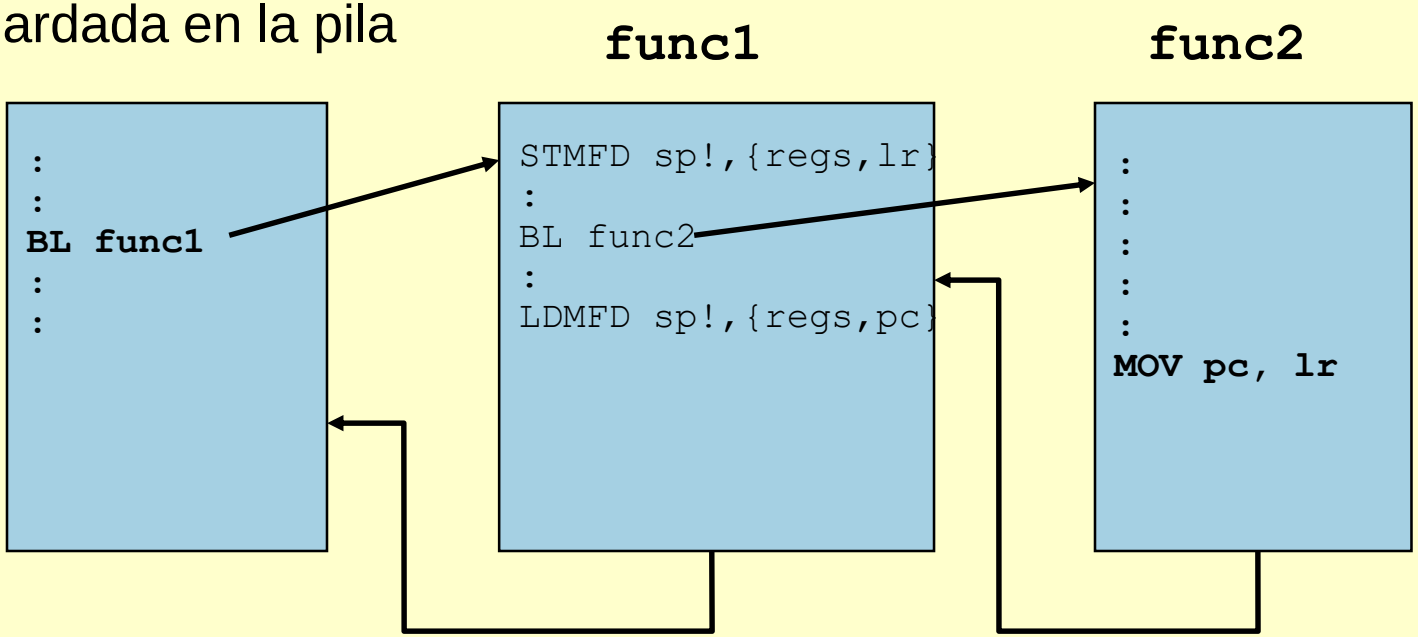
Condicional

```
CMP   r0, #0
ADDEQ r1, r1, #1
ADDNE r2, r2, #1
...
```

- 3 instrucciones
- 3 palabras
- 3 ciclos

Branches y subrutinas

- B <etiqueta>
  - Relativo al PC ±32 Mbyte range.
- BL <subrutina>
  - Almacena la dirección de retorno en LR
  - Retorna restaurando el PC desde LR
  - Para lamadas a subrutina desde una subrutina, LR debe ser guardada en la pila



Branches y subrutinas

```
        mov     r4,#0x7c
        ldr     r0,=0x55555555
        ldr     sp,=Data2
        bl      subru1
        mov     r5,r12
        addsr12,pc,lr
subru1
        STMFD   sp!,{r0-r8,lr}
        bl      subru2
        LDMFD   sp!,{r0-r8,pc}
```

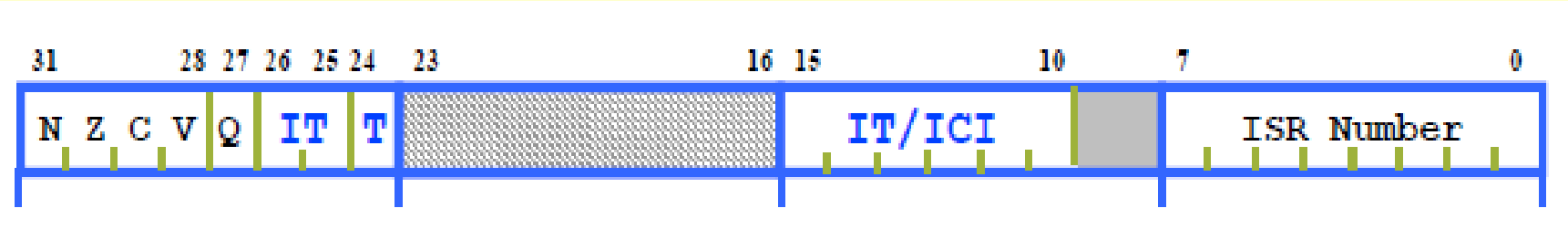
```
subru2  adds r0,r1,r9
        add     r5,r5,#1
        mov     pc,lr

        AREAData2, DATA
Stackspace      40

        end
```

Instrucciones reservadas de control

Acceso PSR - Reservadas



- MRS y MSR permiten transferir CPSR / SPSR de/a un registro o tomar un valor inmediato.
  - MSR permite actualizar todo el registro o una parte del mismo
- Las interrupciones pueden ser habilitadas/deshabilitadas y lcambiar los modos escribiendo al CPSR
  - Típicamente se deben emplear estragias de read/modify/write :

```
MRS r0,CPSR           ; copia el CPSR a r0
BIC r0,r0,#0x10000000; ; limpiar bit V
MSR CPSR,r0           ; Escribir el valor midificado al byte‘c’
```

- En modo usuario sólo pueden modificarse los flags y en nivel privilegiado

Instrucciones de control

- Instrucciones de excepciones: SWI, SoftWare Interrupt.
- Instrucciones del Coprocesador: CDP, LDC, STC, MRC, MCR.
- Cortex no ejecuta estas instrucciones pero deja al coprocesador la manipulación de ellas.

Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Ingeniería Electrónica

Técnicas Digitales II