Instrucciones Lógicas

Resumen de AND, ORR, EOR y BIC

Las instrucciones AND, EOR y ORR realizan operaciones AND, OR exclusivas y OR a nivel de bits en los valores de Rn y Operando 2

La instrucción BIC (Bit Clear) realiza una operación AND en los bits en Rn con los complementos de los bits correspondientes en el valor de Operando 2.

En ciertas circunstancias, el ensamblador puede sustituir BIC por AND, AND por BIC. Tengamos esto en cuenta al leer vuelco de memoria.

Sintaxis General:

```
   op{S}{cond} Rd, Rn, Oper2
```

op es una de las siguientes instrucciones:

- AND (Y lógico).
- ORR (O Lógico).
- EOR (O Lógico exclusivo).
- BIC (Y Lógico negado).

AND

AND realiza un AND lógico a nivel de bit entre dos valores. El primer valor proviene de un registro. El segundo valor puede ser un valor inmediato o un registro, y se le puede aplicar un shift antes de la operación AND. AND puede actualizar opcionalmente los flags en función del resultado

Sintaxis

```
   AND{<cond>}{S}  <Rd>, <Rn>, <shifter_operand>
```

Donde

<cond> Es la condición bajo la cual se ejecuta la instrucción. Si se omite <cond>, se usa la condición AL (siempre).

S Hace que la instrucción actualize el CPSR en base al resultado de la instrucción. Si se omite S, la instrucción no cambia el CPSR.

<Rd> Especifica el registro de destino.

<Rn> Especifica el registro que contiene el primer operando.

<shifter_operand> Especifica el segundo operando. Puede ser un #<immediate> (inmediato de 8 bits), un registro ó un registro con shift

Como se modifican los flags:

```
 if S == 1 then
    N Flag = Rd[31]
    Z Flag = if Rd == 0 then 1 else 0
    C Flag = shifter_carry_out= C Flag anterior (para el caso de
direccionamiento a registro)
    V Flag = no es afectado
```

Uso:

AND se usa para extraer un campo de un registro, haciendo AND al registro con un valor de máscara que tiene 1s en el campo a extraer y 0s en otros lugares.

Ejemplo:

```
   .data
   valor: .word 0xaaaaaaaa
   mascara: .word 0xf0000000
   .text
   .global main
   main:
    ldr r1, =valor
    ldr r1, [r1]        //r1=0xaaaaaaaa
    ldr r0, =mascara
    ldr r0, [r0]         // r0=0xf0000000
    ands r2, r1, r0        //r2=r1 and r0 = 0xa0000000
                //bits NZCV de cpsr pasan de 0x6 (0110) a 0xa (1010)
   fin:
    mov r7, #1        // Salida al sistema
    swi 0
```

Ejercicio: Repetir el código anterior partiendo de un registro cpsr con los bits NZCV en 0. Verificar que el valor del carry flag C se mantiene después de la instrucción AND. Modificar la mascara para extraer del registro r1 los dos bytes mas significativos y los dos bits menos significativos.

ORR

ORR (OR lógico) realiza un OR bit a bit (inclusive) entre dos valores. El primer valor proviene de un registro. El segundo valor puede ser un valor inmediato, un registro ó un registro con shift. ORR puede actualizar opcionalmente los flags en función del resultado

Sintaxis

```
   ORR{<cond>}{S}  <Rd>, <Rn>, <shifter_operand>
```

Donde los operandos y los componentes opcionales de la instruccion se comportan como en la instrucción AND. En particular:

<shifter_operand> Especifica el segundo operando. Puede ser un #<immediate> (inmediato de 8

bits), un registro ó un registro con shift

Los flags se modifican igual que en la Instruccion AND.

Uso: Se usa ORR para setear los bits seleccionados en un registro. Para cada bit, OR con 1 setea el bit, y OR con 0 lo deja sin cambios

Ejemplo:

```
    orrs r2, r1, r0        //r2=r1 or r0
```

Ejercicio: Reciclar el ejemplo de AND para setear en 1 los 8 bits más significativos del registro r1 y almacenarlo en r2. Verificar los valores intermedios con gdb.

EOR

EOR (OR exclusivo) realiza un OR exclusivo bit a bit entre dos valores. El primer valor proviene de un registro. El segundo valor puede ser un valor inmediato, un registro ó un registro con shift. EOR puede actualizar opcionalmente los flags en función del resultado

Sintaxis

```
   EOR{<cond>}{S}  <Rd>, <Rn>, <shifter_operand>
```

Donde los operandos y los componentes opcionales de la instrucción se comportan como en la instrucción AND. Los flags se modifican igual que en la instrucción AND.

Uso: Se usa EOR para invertir los bits seleccionados en un registro. Para cada bit, EOR con 1 invierte ese bit, y EOR con 0 lo deja sin cambios.

Se puede usar para encriptar el contenido de un registro. Para desencriptar el contenido del registro basta aplicar xor nuevamente.

Ejemplo:

```
    .data
    valor: .word 0x12345678
    clave: .word 0xf0f0f0f0
    .text
    .global main
    main:
        ldr r1, =valor
        ldr r1, [r1]        //r1=0x12345678
        ldr r0, =clave
        ldr r0, [r0]         // r0=0xf0f0f0f0
        eors r2, r1, r0        //r2 = r1 xor r0 = 0xe2c4a688 (r1 encriptado)
        eors r3, r2, r0         //r3 = r1
    fin:
        mov r7, #1        // Salida al sistema
        swi 0
```

Ejercicio: Modificar el ejemplo anterior para invertir los 8 bits más significativos y los 8 bits menos significativos del registro r1 y almacenarlo en r2. Verificar que aplicando xor se recupera el valor original de r1.

BIC

BIC (Bit Clear) realiza un AND a nivel de bit entre un valor con el complemento de un segundo valor. El primer valor proviene de un registro. El segundo valor puede ser un valor inmediato, un registro ó un registro con shift. BIC puede actualizar opcionalmente los flags en función del resultado

Sintaxis

```
BIC{<cond>}{S}  <Rd>, <Rn>, <shifter_operand>
```

Donde los operandos y los componentes opcionales de la instrucción se comportan como en la instrucción AND. Los flags se modifican igual que en la instrucción AND.

Uso:

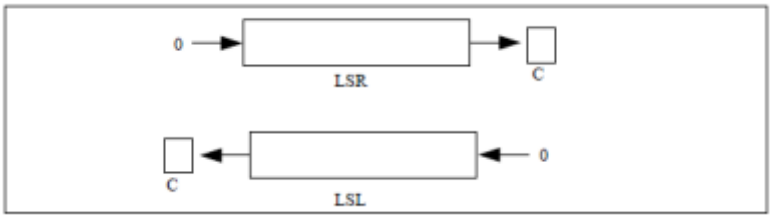
Se usa BIC para borrar los bits seleccionados en un registro. Para cada bit, BIC con 1 borra el bit y BIC con 0 lo deja sin cambios.

Ejemplo:

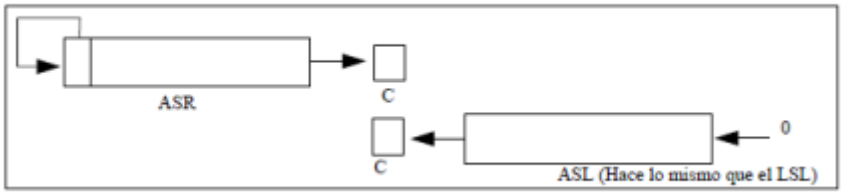
```
    .data
    valor:  .word 0x12345678
    borrar: .word 0xff0000ff
    .text
    .global main
    main:
        ldr r1, =valor
        ldr r1, [r1]        //r1=0x12345678
        ldr r0, =borrar
        ldr r0, [r0]         // r0=0xff0000ff
        bics r2, r1, r0        //r2 = borrar en r1 los bits marcados en r0
                             //r2 = 0x00345600
    fin:
        mov r7, #1        // Salida al sistema
        swi 0
```

Rotaciones y desplazamientos

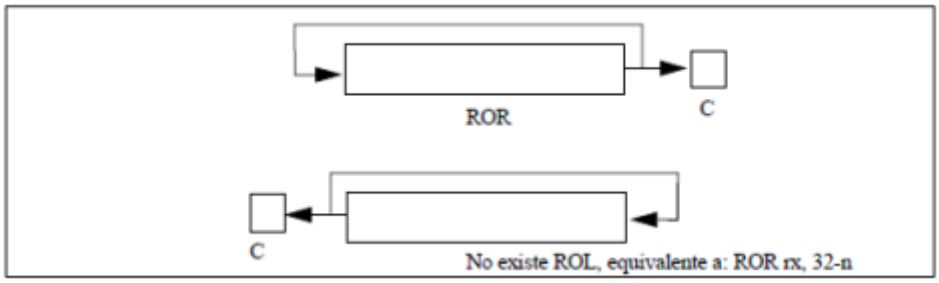
Veremos el funcionamiento de las instrucciones de desplazamiento y rotación. Las instrucciones de desplazamiento pueden ser lógicas o aritméticas. Los desplazamientos lógicos desplazan los bit del registro fuente introduciendo ceros (uno o más de uno). El último bit que sale del registro fuente se almacena en el flag C (en la siguiente figura).



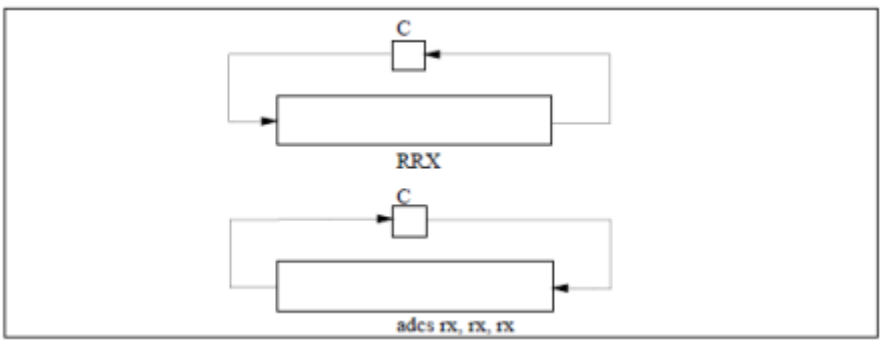
El desplazamiento aritmético hace lo mismo, pero manteniendo el signo. Veamos la siguiente figura



Las instrucciones de rotación también desplazan, pero el bit que sale del valor se realimenta. No existe ninguna instrucción para rotar hacia la izquierda ROL, ya que puede simularse con la de rotación a la derecha **ROR** que sí existe. En estas instrucciones el bit desplazado fuera es el mismo que el que entra, además de dejar una copia en el flag C (Ver abajo).



Las instrucciones de rotación con el carry funcionan de manera similar, pero el bit que entra es el que había en el flag C y el que sale va a parar al flag C. Estas instrucciones sólo rotan un bit, al contrario que las anteriores que podían rotar/desplazar varios. La rotación con carry a la derecha es RRX, no existe la contrapartida RLX porque se puede sintetizar con otra instrucción ya existente adcs. Con adcs podemos sumar un registro consigo mismo, que es lo mismo que multiplicar por 2 o desplazar 1 bit hacia la izquierda. Si a esto le añadimos el bit de carry como entrada y actualizamos los flags a la salida, tendremos exactamente el mismo comportamiento que tendría la instrucción RLX.



From:
http://wiki.educabit.ar/ - **Wiki Sistemas**

Permanent link:
http://wiki.educabit.ar/doku.php?id=arm_inst_logicas

Last update: **2025/09/11 22:48**