

Figura 1.2. Registro de Estado del ARM (CPSR)

Tabla 1.1. Descripción del significado de los bits de condición

Bit	Significado
N	Indica si la última operación dio como resultado un valor negativo (N = 1) o positivo (N = 0)
Z	Indica si el resultado de la última operación fue cero (Z = 1)
C	Su valor depende del tipo de operación: <ul style="list-style-type: none"> - Para suma o comparación, C = 1 si hubo <i>carry</i> - Para las operaciones de desplazamiento toma el valor del bit saliente
V	En el caso de una suma o una resta V = 1 indica que hubo <i>overflow</i>

1.4 Repertorio de instrucciones

Dividiremos las instrucciones del repertorio en cuatro grupos:

- Aritmético-lógicas
- Multiplicación
- Acceso a memoria
- Salto

1.3.1 Instrucciones aritmeticológicas:

En este apartado veremos las instrucciones que usan la unidad aritmético-lógica (ALU) del procesador. No incluimos las instrucciones de multiplicación, que veremos más adelante, ya que se ejecutan en otro módulo. En general, además de escribir el resultado en el registro destino cualquier instrucción puede modificar los flags de CPSR si se le añade como sufijo una S.

La sintaxis de las instrucciones aritmético-lógicas es la siguiente:

Instrucción{S} Rd, Rn1, Rn2

Rd <- Rn1 Operación Rn2



Tabla 1.2: Instrucciones aritmeticológicas más comunes

Mnemo	Operación	Acción
AND	AND Lógica	$Rd \leftarrow Rn1 \text{ AND } Rn2$
ORR	OR Lógica	$Rd \leftarrow Rn1 \text{ OR } Rn2$
EOR	XOR Lógica	$Rd \leftarrow Rn1 \text{ EOR } Rn2$
ADD	Suma	$Rd \leftarrow Rn1 + Rn2$
SUB	Resta	$Rd \leftarrow Rn1 - Rn2$
RSB	Resta inversa	$Rd \leftarrow Rn2 - Rn1$
ADC	Suma con acarreo	$Rd \leftarrow Rn1 + Rn2 + \text{Carry Flag}$
SBC	Resta con acarreo	$Rd \leftarrow Rn1 - Rn2 - \text{NOT}(\text{Carry Flag})$
RSC	Resta inversa con acarreo	$Rd \leftarrow Rn2 - Rn1 - \text{NOT}(\text{Carry Flag})$
CMP	Comparar	Hace $Rn1 - Rn2$ y actualiza los flags de CP-SR convenientemente
MOV	Mover a un registro	$Rd \leftarrow Rn2$

Donde:

- Instrucción: alguno de los mnemotécnicos de la Tabla 1.2
- S: si se incluye este campo la instrucción modifica los indicadores (flags) de condición de CPSR.
- Rd: registro destino (donde se almacena el resultado).
- Rn1: registro fuente (primer operando). Para todas las instrucciones menos MOV.
- Rn2: segundo operando, normalmente se denomina shifter_operand. Puede ser un registro (Ri) o un valor constante (#N).

Tabla 1.3. Ejemplos de instrucciones aritmeticológicas

ADD R0, R1, #1	$R0 = R1 + 1$
ADD R0, R1, R2	$R0 = R1 + R2$
MOV R10, #5	$R10 = 5$
SUB R0, R2, R3	$R0 = R2 - R3$
SUBS R0, R2, R3	$R0 = R2 - R3$ y actualiza N, Z, C y V convenientemente
CMP R4, R5	Compara R4 con R5 actualizando N y Z convenientemente

1.3.2 Instrucciones de multiplicación:

Hay diversas variantes de la instrucción de multiplicación debido a que al multiplicar dos datos de n bits necesitamos $2n$ bits para representar correctamente cualquier resultado, y a que se dispone de multiplicaciones con y sin signo. Las principales variantes se describen en la Tabla 1.4. Todas las instrucciones pueden modificar opcionalmente los bits Z y N del registro de estado en función de que el resultado sea cero o negativo si se les pone el sufijo S. Los operandos siempre están en registros y el resultado se almacena también en uno o dos registros, en función del tamaño deseado (32 o 64 bits).



Tabla 1.4. Instrucciones de multiplicación

Mnemotécnico	Operación
MUL Rd, Rm, Rs	Multiplicación, resultado 32 bits: $Rd \leftarrow (Rm * Rs) [31..0]$
MLA Rd, Rm, Rs, Rn	Multiplicación más suma, 32 bits: $Rd \leftarrow (Rm * Rs + Rn) [31..0]$
SMULL RdLo, RdHi, Rm, Rs	Multiplicación con signo (C2), resultado 64 bits: $RdHi \leftarrow (Rm * Rs) [63..32]; RdLo \leftarrow (Rm * Rs) [31..0]$
UMULL RdLo, RdHi, Rm, Rs	Multiplicación sin signo (binario puro), 64 bits: $RdHi \leftarrow (Rm * Rs) [63..32]; RdLo \leftarrow (Rm * Rs) [31..0]$

Tabla 1.5. Ejemplos de instrucciones de multiplicación

MUL R0, R1, R2	$R0 = R1 \times R2$
MLA R0, R1, R2, R3	$R0 = (R1 \times R2) + R3$
MULS R5, R8, R9	$R5 = R8 \times R9$ y actualiza N, Z, C y V convenientemente
UMULL R0, R1, R2, R3	$R0 = R2 \times R3 [31 \dots 0]$ y $R1 = R2 \times R3 [63 \dots 32]$

1.3.3 Instrucciones de acceso a memoria (load y store):

Las instrucciones de load (LDR) se utilizan para cargar un dato de memoria sobre un registro. Las instrucciones de store (STR) realizan la operación contraria, copian en memoria el contenido de un registro.

Para formar la dirección de memoria a la que se desea acceder, se utiliza un registro base y un desplazamiento (offset). Este último puede ser un inmediato, otro registro o un registro desplazado. Hay tres variantes o tres formas de combinar el registro base y el desplazamiento, en esta primera práctica solo explicaremos la más común:

Indirecto de registro con desplazamiento: la dirección de memoria a la que se desea acceder se obtiene al sumar al desplazamiento el dato contenido en el registro base.

LDR Rd, [Rb, #desp]	$Rd \leftarrow \text{Memoria}(Rb + \text{desp})$
STR Rf, [Rb, #desp]	$\text{Memoria}(Rb + \text{desp}) \leftarrow Rf$

Tabla 1.6. Ejemplos de instrucciones de almacenamiento

LDR R0, [R1]	$R0 = \text{Mem}(R1)$ Almacena en R0 lo que se encuentra en memoria almacenado en la dirección que contiene R1
LDR R0, [R1, #7]	$R0 = \text{Mem}(R1 + 7)$
STR R5, [R8, #-4]	$\text{Mem}(R8 - 4) = R5$ Almacena en la dirección de memoria formada por $(R8 - 4)$ el dato almacenado en R5
STR R5, [R8, #0x010]	$\text{Mem}(R8 + 16) = R5$ Todo número 0xNNN es un número representado en hexadecimal



1.3.4 Instrucciones de salto:

La instrucción explícita de salto es Branch (B). Realiza un salto relativo al PC, hacia atrás o hacia delante. La distancia a saltar está limitada por los 24 bits con los que se puede codificar el desplazamiento dentro de la instrucción (operando inmediato). La sintaxis es la siguiente:

B{Condicion} Desplazamiento

PC <- PC + Desplazamiento

donde Condición indica una de las condiciones de la tabla 1.7 (EQ,NE,GE,GT,LE,LT,. . .) y Desplazamiento es un valor inmediato con signo que representa un desplazamiento respecto del PC.

La dirección a la que se salta es codificada como un desplazamiento relativo al PC (la instrucción a la que queremos saltar se encuentra N instrucciones por delante o por detrás de la instrucción actual). Sin embargo, a la hora de programar en ensamblador utilizaremos etiquetas y será el *enlazador* el encargado de calcular el valor exacto del desplazamiento.

Los saltos condicionales se ejecutan solamente si se cumple la condición del salto. Una instrucción anterior tiene que haber activado los indicadores de condición del registro de estado. Normalmente esa instrucción anterior es CMP (ver instrucciones aritméticas), pero puede ser cualquier instrucción con sufijo S.

Tabla 1.7. Condiciones asociadas a las instrucciones de salto

Mnemotécnico	Descripción	Flags
EQ	Igual	Z=1
NE	Distinto	Z=0
HI	Mayor que (sin signo)	C=1 & Z=0
LS	Menor o igual que (sin signo)	C=0 or Z=1
GE	Mayor o igual que (con signo)	N=V
LT	Menor que con signo	N!=V
GT	Mayor que con signo	Z=0 & N=V
LE	Menor o igual que (con signo)	Z=1 or N!=V
(vacío)	Siempre (incondicional)	

En el caso de que no se cumpla la condición, el flujo natural del programa se mantiene, es decir, se ejecuta la instrucción siguiente a la del salto.

Tabla 1.8. Ejemplos de instrucciones de salto condicional

B etiqueta	Salta siempre a la dirección dada por la etiqueta
BEQ etiqueta	Salta a la etiqueta si el flag Z está activado
BLS etiqueta	Salta a la etiqueta si el flag Z o el N están activados
BHI etiqueta	Salta a etiqueta si C =1 y Z = 0



Tabla 1.9. Traducción de lenguaje de alto nivel a lenguaje ensamblador

if (R1 > R2) R3 = R4 + R5; else R3 = R4 - R5 sigue la ejecución normal	CMP R1, R2 BLE else ADD R3, R4, R5 B fin_if else: SUB R3, R4, R5 fin_if: <i>sigue la ejecución normal</i>
for (i=0; i<8; i++) { R3 = R1 + R2; } sigue la ejecución normal	MOV R0, #0 @R0 actúa como índice i for: CMP R0, #8 BGE fin_for ADD R3, R1, R2 ADD R0, #1 B for fin_for: <i>sigue la ejecución normal</i>
repeat until i=8 { R3 = R1 + R2; } sigue la ejecución normal	MOV R0, #0 @R0 actúa como índice i repeat: ADD R3, R1, R2 ADD R0, #1 CMP R0, #8 BNE repeat <i>sigue la ejecución normal</i>
while (R1 < R2) { R2= R2-R3; } sigue la ejecución normal	while: CMP R1, R2 BGE fin_w SUB R2, R2, R3 B while fin_w: <i>sigue la ejecución normal</i>

1.5 Estructura de un programa en lenguaje ensamblador

Para explicar la estructura de un programa en lenguaje ensamblador y las distintas directivas del ensamblador utilizaremos como guía el sencillo programa descrito en el cuadro 1.1.

Lo primero que podemos ver en el listado del cuadro 1.1 es que un programa en lenguaje ensamblador no es más que un texto estructurado en líneas con el siguiente formato:

etiqueta: <instrucción o directiva> @ comentario



Dentro del fichero que define el programa hay una sección (líneas contiguas) dedicadas a definir dónde se van a almacenar los datos, ya sean datos de entrada o de salida. Y otra sección dedicada a describir el algoritmo mediante instrucciones de código ensamblador. Como el ARM es una máquina Von Neuman los datos y las instrucciones utilizan el mismo espacio de memoria. Como programa informático (aunque esté escrito en lenguaje ensamblador), los datos de entrada estarán definidos en unas direcciones de memoria, y los datos de salida se escribirán en direcciones de memoria reservadas para ese fin. De esa manera si se desean cambiar los valores de entrada al programa se tendrán que cambiar a mano los valores de entrada escritos en el fichero. Para comprobar que el programa funciona correctamente se tendrá que comprobar los valores almacenados en las posiciones de memoria reservadas para la salida una vez se haya ejecutado el programa.

```
.global start

.data
.equ UNO, 0x01
DOS: .word 0x02

.bss
RES: .space 4

.text
start:
    MOV R0, #UNO
    LDR R1, =DOS
    LDR R2,[R1]
    ADD R3, R0, R2
    LDR R4, =RES
    STR R3, [R4]
FIN:  B .
    .end
```

Cuadro 1.1. Programa en lenguaje ensamblador

Los términos utilizados en la descripción de la línea son:

etiqueta: (opcional) es una cadena de texto que el ensamblador relacionará con la dirección de memoria correspondiente a ese punto del programa. Si en cualquier otro punto del programa se utiliza esta cadena en un lugar donde debiese ir una dirección, el ensamblador sustituirá la etiqueta por el modo de acceso correcto a la dirección que corresponde a la etiqueta. Por ejemplo, en el programa del cuadro 1 la instrucción `LDR R1, =DOS` carga en el registro R1 la posición de memoria (dirección) donde se encuentra almacenada la variable DOS, actuando a partir de este momento el registro R1 como si fuera un puntero. El ensamblador es lo suficientemente inteligente como para codificar correctamente esta instrucción, calculando automáticamente la posición de la

variable DOS. Si se desea acceder al valor de la variable DOS se deberá escribir LDR R2, [R1], siempre que R1 almacene el puntero a DOS.

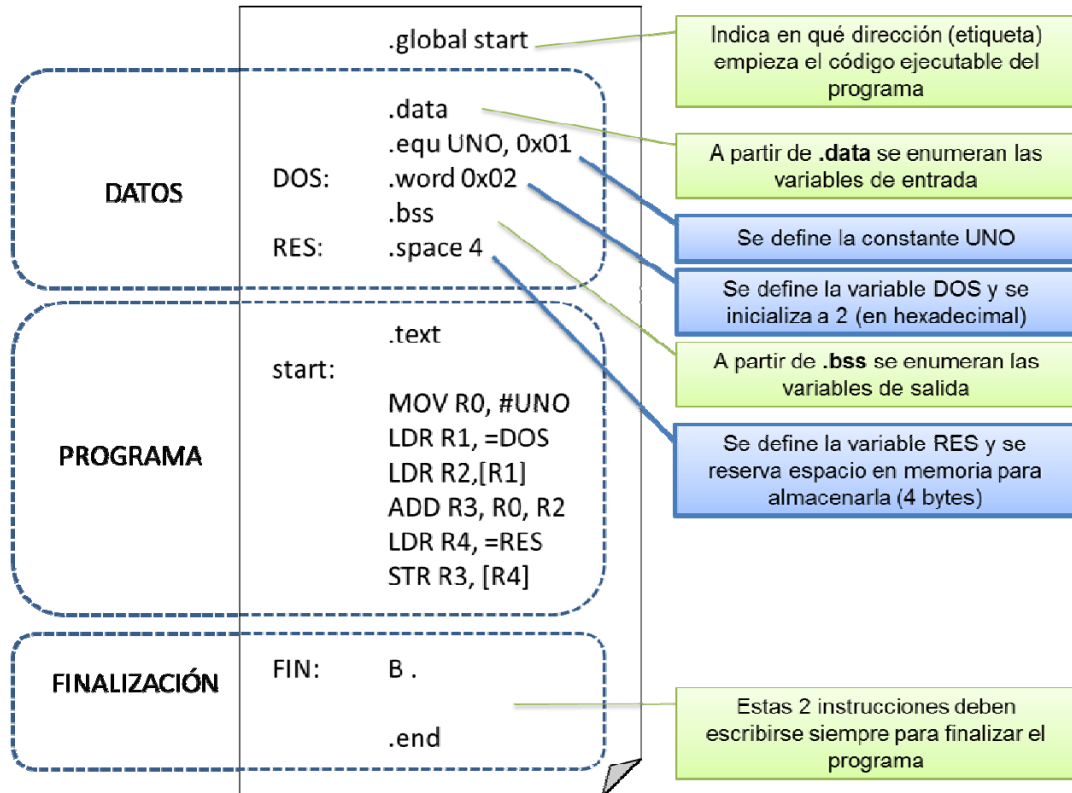


Figura 1.3 Descripción de la estructura de un programa en ensamblador: datos

- **instrucción**: el mnemotécnico de una instrucción de la arquitectura destino (alguna de las descritas en la sección 1.4, ver Figura 1.4). A veces puede estar modificado por el uso de etiquetas o macros del ensamblador que faciliten la codificación. Un ejemplo es el caso descrito en el punto anterior donde la dirección de un load se indica mediante una etiqueta y es el ensamblador el que codifica esta dirección como un registro base más un desplazamiento.
- **directiva**: es una orden al propio programa ensamblador. Las directivas permiten inicializar posiciones de memoria con un valor determinado, definir símbolos que hagan más legible el programa, marcar el inicio y el final del programa, etc (Ver Figura 1.3). Las directivas también se utilizan para inicializar variables. Aparte de escribir el código del algoritmo mediante instrucciones de ensamblador, el programador en lenguaje ensamblador debe reservar espacio en memoria para las variables, y en caso de que deban tener un valor inicial, escribir este valor en la dirección correspondiente.

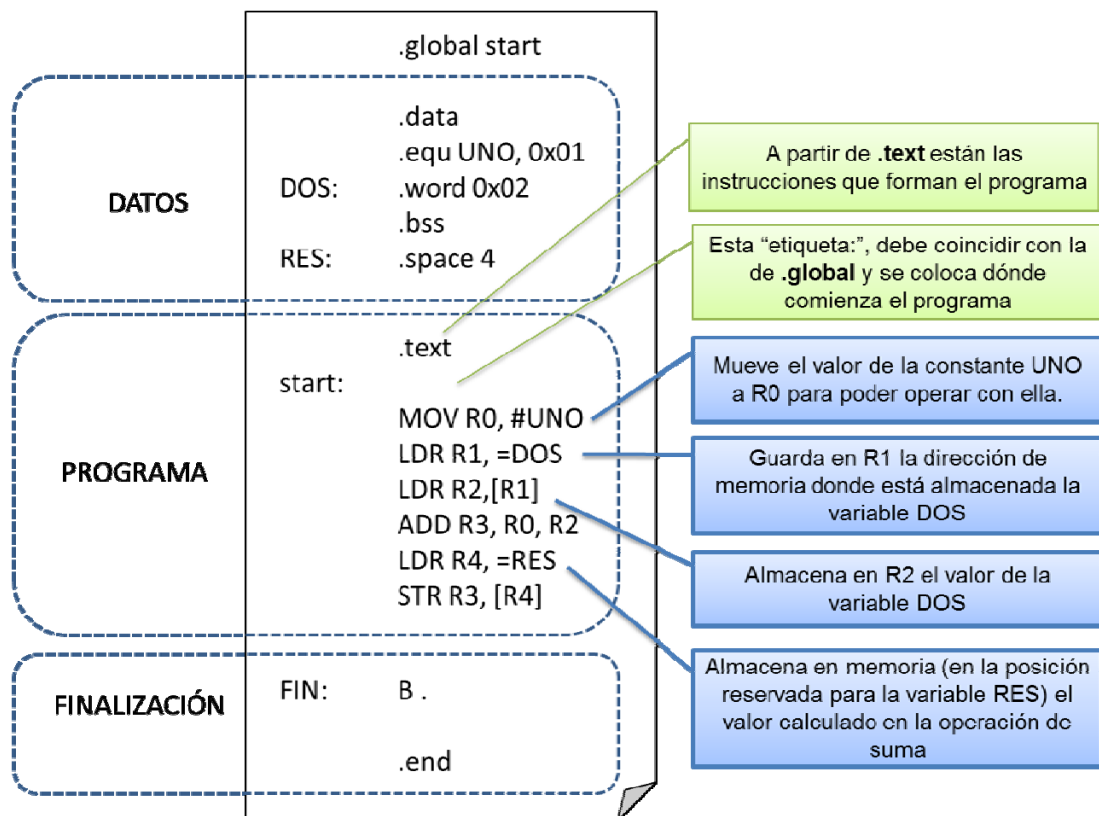


Figura 1.4 Descripción de la estructura de un programa en ensamblador: instrucciones

- **.global**: exporta un símbolo para otros ficheros en la etapa de enlazado. El comienzo del programa se indica mediante la directiva **.global** etiqueta (**.global** **start** en el ejemplo del cuadro 1.1), dicha etiqueta aparecerá otra vez justo antes de la primera instrucción del programa, para indicar dónde se encuentra la primera instrucción ejecutable por el procesador.
- **.equ**: se escribe para poder utilizar símbolos, constantes. De forma sencilla podemos entender un símbolo como una cadena de caracteres que será sustituida allí donde aparezca por un valor, que nosotros definimos. Por ejemplo, **.equ** **UNO**, **0x01** define un símbolo **UNO** con valor **0x01**. Así, cuando en la línea **MOV R0, #UNO** se utiliza el símbolo, el ensamblador lo sustituirá por su valor.
- **.word**: se suele utilizar para inicializar las variables de entrada al programa. Inicializa la posición de memoria actual con el valor indicado tamaño palabra (también podría utilizarse **.byte**). Por ejemplo: **DOS .word 0x02**, inicializa la posición de memoria con etiqueta **DOS** al valor 2 donde **0x** indica que es hexadecimal. .
- **.space**: reserva espacio en memoria tamaño byte para guardar las variables de salida, si éstas no se corresponden con las variables de entrada. Siempre es necesario indicar el espacio que se quiere reservar. Por ejemplo, **RES: .space 4** se reservan cuatro bytes (una palabra (**word**)) que quedan sin inicializar. La etiqueta **RES** podrá utilizarse en el resto del programa para referirse a la dirección correspondiente a esta palabra.



- **.end:** Finalmente, el ensamblador dará por concluido el programa cuando encuentre la directiva .end. El texto situado detrás de esta directiva de ensamblado será ignorado.
- **secciones:** normalmente el programa se estructura en secciones (generalmente .text, .data y .bss). Para definir estas secciones simplemente tenemos que poner una línea con el nombre de la sección. A partir de ese momento el ensamblador considerará que debe colocar el contenido subsiguiente en la sección con dicho nombre.
 - **.bss:** es la sección en la que se reserva espacio para almacenar el resultado.
 - **.data:** es la sección que se utiliza para declarar las variables con valor inicial
 - **.text:** contiene el código del programa.
- **comentario:** (opcional) una cadena de texto para comentar el código. Con @ se comenta hasta el final de la línea actual. Pueden escribirse comentarios de múltiples líneas como comentarios C (entre /* y */).

Los pasos seguidos por el entorno de compilación sobre el código presentado en el Cuadro 1.1 se resumen en la siguiente tabla.

Tabla 1.10. Traducción de código ensamblador a binario

start: MOV R0, #UNO LDR R1, =DOS LDR R2,[R1] ADD R3, R0, R2 LDR R4, =RES STR R3, [R4] FIN: B . .end	mov r0, #1 ldr r1, [pc, #10] ldr r2, [r1] add r3, r0, r2 ldr r4, [pc, #8] str r3, [r4] b 0xc00001c	00000001000000001010000011100111 00010000000100001001111111100101 00000000001000001001000111100101 00000010001100001000000011100000 00001000010000001000111111100101 00000000001100001000010011100101 11111101111111111111111111101010
---	--	--

En la Tabla 1.10 el código de la primera columna se corresponde con el código en lenguaje ensamblador con etiquetas, este código nos facilita una escritura rápida del algoritmo sin tener que realizar cálculos relativos a la posición de las variables en memoria para realizar su carga en el registro asignado a dicha variable. En la columna del centro aparece el código ensamblador que ejecuta el procesador, en este código las etiquetas se han traducido por un valor relativo al contador de programa (el dato se encuentra X posiciones por encima o por debajo de la instrucción actual). Además, las pseudo-instrucciones (instrucciones que no pertenecen al código máquina de la arquitectura ARM), se traducen por la instrucción o instrucciones correspondientes. Una vez obtenido este código desambiguado ya se puede realizar una traducción directa a ceros y unos, que es el único lenguaje que entiende el procesador, esa traducción está presente en la columna 3, donde por ejemplo los últimos bits se corresponden con el código de operación de cada una de las instrucciones (se puede comprobar con las tres instrucciones ldr).