Universidad Nacional Autónoma de México
Facultad de Ingeniería

Microprocesadores y Microcontroladores

Tema 3

Modos de direccionamiento y conjunto de instrucciones

M. I. Christo Aldair Lara Tenorio

2025-1

Tabla de contenidos

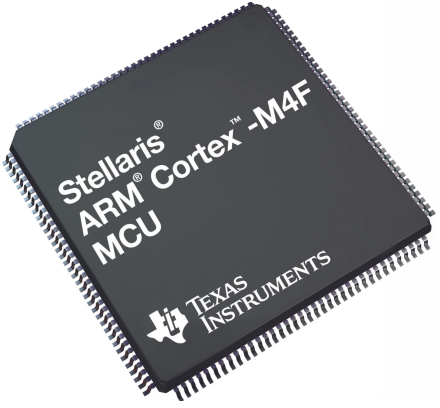
Objetivo del tema

Objetivo general:
El alumno conocerá las formas de búsqueda de operandos y el conjunto de instrucciones.

Contenido:

3.1. Modos de direccionamiento.

3.2. Conjunto de instrucciones.



Instrucción

Operación específica que puede ser ejecutada por un procesador.

- Indica la operación que realizará el procesador.
- Cumplen con un formato específico (sintaxis).
- Pertenecen al conjunto de instrucciones (ISA, del inglés *instruction set architecture*) del procesador.
- Las instrucciones están codificadas en lenguaje máquina (0 y 1).

Codificación	Hexadecimal	Instrucción
0000 0000	0x00	OR
0000 0001	0x01	AND
0000 0010	0x02	XOR
0000 0011	0x03	Suma
0000 0100	0x04	Resta
0000 0101	0x05	NOT
0000 0110	0x06	NOP
0000 0111	0x07	Carga

Formato de una instrucción

- Código de operación (Opcode) – Operación a realizar.
- Operandos – Datos a operar.
- Modos de direccionamiento – Ubicación de los operandos.

Sintaxis de una instrucción en lenguaje ensamblador

Las instrucciones en lenguaje ensamblador tienen 4 campos separados por espacios.

```
hola    MOV R0, #100    ; R0 = #100
```

En donde:

- Etiqueta
  Específica la posición en memoria (opcional).
- Opcode
  Específica el comando o instrucción a ejecutar.
- Operandos
  Específica la ubicación del dato que se va a operar.
- Comentarios
  Caracteres que serán ignorados por el procesador (opcional).

Instrucción MOV

Copiar el contenido de un lugar (fuente) a otro (destino).

```
MOV Rd, Op2
```

En donde:

- `Rd` → Registro o ubicación de memoria en donde se cargará el dato copiado.
- `Op2` → Registro, dato o ubicación de memoria desde donde se obtiene el dato a copiar.

Descripción: Copiar el dato de `Op2` en `Rd`.

Características principales:

- No realiza operaciones aritméticas o lógicas en el dato copiado.
- Es bastante útil para copiar datos entre registros o inicializar registros con un dato constante.

Instrucción LDR (Load Register)
Cargar (copiar) datos desde la memoria a un registro.

```
LDR Rd, [Rn]
```

En donde:

- `Rd` → Registro en donde se cargará el dato copiado.
- `Rn` → Dirección de memoria desde donde se va a copiar el dato (apuntador).

Descripción: Copiar el dato de memoria apuntado por `Rn` en `Rd`.

Características principales:

- Permite mover datos desde la memoria a los registros para utilizarlos en cálculos.
- El apuntador se coloca entre corchetes [ ].

Instrucción B (Branch)
Realizar un salto incondicional a otra zona del programa, especificada por una etiqueta.

```
B label
```

En donde:

- `label` → Etiqueta que hace referencia a una dirección de memoria.

Descripción: Saltar a la dirección de memoria referenciada por la etiqueta `label`.

Características principales:

- Permite cambiar la secuencia de ejecución.
- Permite implementar bucles y estructuras de control.
- Útil para controlar el flujo del programa.

Modos de direccionamiento

Formato de una instrucción para determinar el método de acceso a los operandos (datos) necesarios para que el procesador ejecute una instrucción.

- Agregan flexibilidad al modo de acceso a los datos y cómo estos son manipulados en memoria.

Diferentes modos de direccionamiento

- Modo de direccionamiento inmediato.
- Modo de direccionamiento inmediato de registro.
- Modo de direccionamiento de registro indirecto.
- Modo de direccionamiento inmediato indexado.
  - Inmediato pre-indexado.
  - Inmediato post-indexado.
- Modo de direccionamiento relativo del PC.

¡ Manejar los diferentes modos de direccionamiento permite programar eficientemente !

Modo de direccionamiento inmediato
El operando específica directamente el operando dentro de la misma instrucción.

```
MOV R0, #100
```

Descripción: Cargar el dato constante `#100` en el registro `R0`.

Uso típico: Almacenar constantes en registros.

Ejemplo: Le pides indicaciones a un taxista para llegar a un negocio y como buen conocedor de la ciudad te da la dirección exacta.

Modo de direccionamiento inmediato de registro
El operando está almacenado en un registro, por lo que la instrucción indica el registro que contiene al operando.

```
MOV R0, R1
```

Descripción: Cargar el dato almacenado en el registro `R1` en el registro `R0`.

Uso típico: Realizar operaciones en donde los operandos ya se encuentran almacenados en registros.

Ejemplo: Le pides indicaciones a un taxista para llegar a un negocio, quien no sabe exactamente cómo llegar pero te entrega un mapa para llegar a tu destino.

Modo de direccionamiento de registro indirecto
El operando se encuentra almacenado en memoria cuya dirección está almacenada en un registro, por lo que la instrucción indica el registro que apunta a la localidad de memoria.

```
LDR R0, [R1]
```

Descripción: Leer de memoria el dato apuntado por `R1` y almacenarlo en `R0`.

Uso típico: Acceder a memoria (leer o escribir).

Ejemplo: Le pides indicaciones a un taxista para llegar a un negocio, pero al ser nuevo no conoce bien la ciudad, así que te da el número de un colega suyo con mayor experiencia quien te da la dirección exacta.

Modo de direccionamiento inmediato indexado

Un registro apunta a una dirección base, a la cual se le suma/resta un offset para obtener la dirección efectiva del operando.

Inmediato pre-indexado

El offset se suma/resta antes de usar la dirección de apuntamiento.

```
LDR R0, [R1, #4] (a)
LDR R0, [R1, #4]! (b)
```

Descripción: Leer de memoria el dato apuntado por `R1 + 4` y almacenarlo en `R0`, sin alterar (a) o actualizando (b) el valor almacenado en `R1`.

Inmediato post-indexado

El offset se suma/resta después de usar la dirección de apuntamiento.

```
LDR R0, [R1], #4
```

Descripción: Leer de memoria el dato apuntado por `R1` y almacenarlo en `R0`, después sumar #4 al valor almacenado en `R1`.

Uso típico: Acceder a arreglos o estructuras almacenados en memoria (leer o escribir).

Modo de direccionamiento relativo del PC
La dirección efectiva es calculada al sumar un offset al valor actual del PC.

```
B label
```

Descripción: Saltar a la dirección de memoria referenciada por `label` (el offset que se suma al PC es relativo, dependiendo de su cuenta actual).

Uso típico: Modificar el flujo del programa al saltar instrucciones y acceder a datos cercanos a la instrucción actual.

Ejemplo de modos de direccionamiento

Ejemplo:

Considerando el siguiente estado de la memoria de instrucciones, ¿qué datos se cargarán en los registros `R0`, y `R1`, después de ejecutar instrucciones con diferentes modos de direccionamiento?

Memoria de datos

Dirección	Dato
0x2000 0000	0x1111 1111
0x2000 0004	0x2222 2222
0x2000 0008	0x3333 3333
0x2000 000C	0x4444 4444
0x2000 0010	0x5555 5555
0x2000 0014	0x6666 6666
0x2000 0018	0x7777 7777
0x2000 001C	0x8888 8888
0x2000 0020	0x9999 9999
0x2000 0024	0xAAAA AAAA
0x2000 0028	0xBBBB BBBB
0x2000 002C	0xCCCC CCCC
0x2000 0030	0xDDDD DDDD

Si al inicio del código `R7 = 0x2000 0008` ...

Direccionamiento	Instrucción	Estado de los registros
Inmediato (registro)	`MOV R0, R7`	`R0 = 0x2000 0008` `R1 = 0x1234 5678`
Indirecto (registro)	`LDR R1, [R0]`	`R0 = 0x2000 0008` `R1 = 0x3333 3333`
Inmediato pre-indexado	`LDR R1, [R0, #8]`	`R0 = 0x2000 0008` `R1 = 0x5555 5555`
Inmediato pre-indexado	`LDR R1, [R0, #8]!`	`R0 = 0x2000 0010` `R1 = 0x5555 5555`
Inmediato post-indexado	`LDR R1, [R0], #8`	`R0 = 0x2000 0018` `R1 = 0x5555 5555`

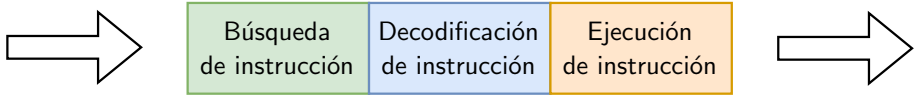
Conjunto de instrucciones

Colección de todas las operaciones básicas que puede realizar un procesador.

Funciones principales:

- Manejo de datos (manipulación de bits).
- Ejecución de operaciones lógicas/aritméticas.
- Lectura/Escritura en memoria.
- Control de procesos.

Proceso de ejecución de las instrucciones en un ARM Cortex-M4:

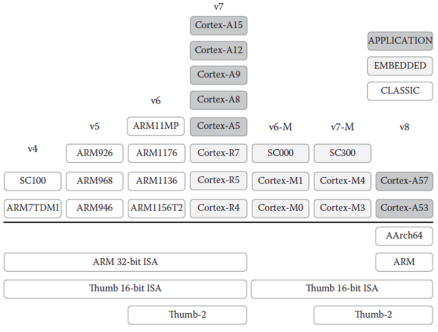


¡ El conjunto de instrucciones de un procesador depende de su arquitectura !

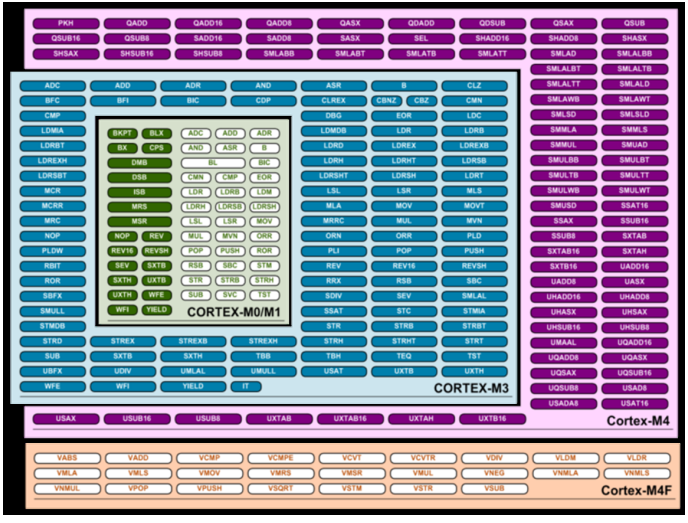
Thumb-2

Tecnología desarrollada por ARM para sus procesadores, como una extensión del conjunto de instrucciones Thumb original.

- Diseñado para mejorar la eficiencia en términos de código compacto y rendimiento.
- Permite utilizar instrucciones de 16 bits (en lugar de 32), reduciendo el tamaño del código.
- Permite combinar instrucciones de 16 y 32 bits en un solo flujo de ejecución, maximizando el rendimiento sin consumir más espacio en memoria.
- Todos los procesadores ARM Cortex-M se basan en la tecnología Thumb-2.
- Ampliamente adoptado en dispositivos móviles y sistemas embebidos (aplicaciones con recursos de memoria y batería limitados).



Instrucciones soportadas por los procesadores Cortex-M



Instrucciones soportadas por los procesadores ARM Cortex-M3/M4F

Los procesadores ARM Cortex-M4 tienen capacidad para ejecutar aproximadamente 203 instrucciones, sin incluir aquellas relaciones con la unidad de punto flotante (FPU).

Clasificación de las instrucciones en 9 grupos, de acuerdo con su función

1. Instrucciones de acceso a memoria.
2. Instrucciones generales de procesamiento de datos.
3. Instrucciones de multiplicación y división.
4. Instrucciones de saturación.
5. Instrucciones de packing y unpacking.
6. Instrucciones de campo de bits.
7. Instrucciones de punto flotante.
8. Instrucciones de salto y control.
9. Instrucciones varias.

Instrucción LDR

Cargar (copiar) datos desde la memoria a un registro.

```
LDR{type}{cond} Rd, [Rn]
```

En donde:

- `Rd` → Registro en donde se cargará el dato copiado.
- `Rn` → Dirección de memoria desde donde se va a copiar el dato (apuntador).

Descripción: Copiar el dato de memoria apuntado por `Rn` en `Rd`.

Características principales:

- Permite mover datos desde la memoria a los registros para utilizarlos en cálculos.
- El apuntador se coloca entre corchetes [ ].

Instrucción LDR

Cargar (copiar) datos desde la memoria a un registro.

```
LDR{type}{cond} Rd, [Rn]
```

En donde:

- `Rd` → Registro en donde se cargará el dato copiado.
- `Rn` → Dirección de memoria desde donde se va a copiar el dato (apuntador).

Sufijo opcional (`type`):

- `B` → Dato de 8 bits sin signo.
- `SB` → Dato de 8 bits con signo.
- `H` → Dato de 16 bits sin signo.
- `SH` → Dato de 16 bits con signo.

Operandos:

```
Rd, [Rn]
Rd, [Rn, #offset]
Rd, [Rn], #offset
Rd, [Rn, #offset]!
Rd, [Rn, <Op2>]
```

<Op2> → Operando flexible.

Instrucción STR

Cargar (copiar) datos desde un registro a memoria.

```
STR{type}{cond} Rd, [Rn]
```

En donde:

- `Rd` → Registro desde donde se va a copiar el dato.
- `Rn` → Dirección de memoria en donde se cargará el dato copiado (apuntador).

Descripción: Copiar el dato del registro `Rd` en la localidad de memoria apuntada por `Rn`.

Características principales:

- Permite mover datos desde los registros a la memoria para almacenarlos.
- El apuntador se coloca entre corchetes [ ].

Instrucción STR

Cargar (copiar) datos desde un registro a memoria.

```
STR{type}{cond} Rd, [Rn]
```

En donde:

- `Rd` → Registro desde donde se va a copiar el dato.
- `Rn` → Dirección de memoria en donde se cargará el dato copiado (apuntador).

Sufijo opcional (`type`):

- `B` → Dato de 8 bits sin signo.
- `H` → Dato de 16 bits sin signo.

Operandos:

```
Rd, [Rn]
Rd, [Rn, #offset]
Rd, [Rn, <Op2>]
```

`<Op2>` → Operando flexible.

Instrucción MOV

Copiar el contenido de un lugar (fuente) a otro (destino).

```
MOV{S}{cond} Rd, <Op2>
```

En donde:

- `Rd` → Registro o ubicación de memoria en donde se cargará el dato copiado.
- `<Op2>` → Operando flexible (registro, constante o ubicación de memoria desde donde se obtiene el dato a copiar).

Descripción: Copiar el dato de `<Op2>` en `Rd`.

Características principales:

- No realiza operaciones aritméticas o lógicas en el dato copiado.
- Es bastante útil para copiar datos entre registros o inicializar registros con un valor constante.

Instrucción MOV

Copiar el contenido de un lugar (fuente) a otro (destino).

```
MOV{S}{cond} Rd, <Op2>
```

En donde:

- `Rd` → Registro o ubicación de memoria en donde se cargará el dato copiado.
- `<Op2>` → Operando flexible (registro, constante o ubicación de memoria desde donde se obtiene el dato a copiar).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, V.

Operandos:

`Rd, <Op2>`

`<Op2>` → Operando flexible*.

* `<Op2>` puede ser una constante de 32 bits, siempre y cuando respete los siguientes formatos:

- 0x00XY00XY
- 0xXY00XY00
- 0xXYXYXYXY

Instrucciones MOVW / MOVT

Copiar 16 bits del contenido de un lugar (fuente) a otro (destino).

```
MOV{W, T}{cond} Rd, #imm16
```

En donde:

- `Rd` → Registro o ubicación de memoria en donde se cargará el dato copiado.
- `#imm16` → Dato de 16 bits.

Descripción: Copiar el dato de `#imm16` en `Rd`.

Características principales:

- No realiza operaciones aritméticas o lógicas en el dato copiado.
- Es bastante útil para copiar datos entre registros o inicializar registros con un valor constante.
- La instrucción `MOVW` realiza la misma función que `MOV`, pero está restringida a operandos de 16 bits.

Instrucciones MOVW / MOVT

Copiar 16 bits del contenido de un lugar (fuente) a otro (destino).

```
MOV{W, T}{cond} Rd, #imm16
```

En donde:

- `Rd` → Registro o ubicación de memoria en donde se cargará el dato copiado.
- `#imm16` → Dato de 16 bits.

Sufijos opcionales:

- `W` → Copiar los 16 bits menos significativos.
- `T` → Copiar los 16 bits más significativos.

Operandos:

`Rd, #imm16`

`#imm16` → Dato de 16 bits.

Operaciones lógicas

El ARM Cortex-M4 puede realizar multiples operaciones lógicas para combinar o extraer información, las cuales toman dos entradas de 32 bits: un registro y un operador flexible (<Op2>), realizando la operación bit a bit.

Por ejemplo $\longrightarrow$ `0x123445678 OR 0x87654321`

R1	0001	0010	0011	0100	0101	0110	0111	1000
<Op2>	1000	0111	0110	0101	0100	0011	0010	0001
ORR	1001	0111	0111	0101	0101	0111	0111	1001

Instrucción AND
AND lógico de 32 bits.

```
AND{S}{cond} {Rd,} Rn, <Op2>
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).

Descripción: Almacenar en `Rd` (o en `Rn`) la AND lógica entre `Rn` y `<Op2>`.

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:
`{Rd,} Rn, <Op2>`
`<Op2>` → Operando flexible.

Instrucción ORR
OR lógico de 32 bits.

```
ORR{S}{cond} {Rd,} Rn, <Op2>
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).

Descripción: Almacenar en `Rd` (o en `Rn`) la OR lógica entre `Rn` y `<Op2>`.

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:
`{Rd,} Rn, <Op2>`
`<Op2>` → Operando flexible.

Instrucción ORN
NOT OR lógico de 32 bits.

```
ORN{S}{cond} {Rd,} Rn, <Op2>
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).

Descripción: Almacenar en `Rd` (o en `Rn`) la OR lógica entre `Rn` y `<Op2>` negado.

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:
`{Rd,} Rn, <Op2>`
`<Op2>` → Operando flexible.

Instrucción EOR
XOR lógico de 32 bits.

```
EOR{S}{cond} {Rd,} Rn, <Op2>
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).

Descripción: Almacenar en `Rd` (o en `Rn`) la XOR lógica entre `Rn` y `<Op2>`.

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:
`{Rd,} Rn, <Op2>`
`<Op2>` → Operando flexible.

Instrucción BIC

Bit clear.

BIC{S}{cond} {Rd,} Rn, <Op2>

En donde:

- Rd → Registro donde se cargará el resultado (opcional).
- Rn → Primer operando (si se omite Rd, almacenará el resultado).
- <Op2> → Segundo operando (flexible).

Descripción: Almacenar en Rd (o en Rn) la AND lógica entre Rn y <Op2> negado, en donde el <Op2> representa los bits que se van a "limpiar".

Sufijos opcionales:

- S → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

{Rd,} Rn, <Op2>

<Op2> → Operando flexible.

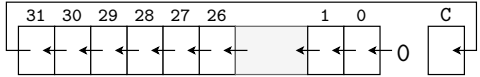
Operaciones de corrimiento/desplazamiento

El ARM Cortex-M4 puede realizar 5 operaciones de corrimiento/desplazamiento para manipulación de bits y optimización de cálculos.

LSR
Logical Shift Right

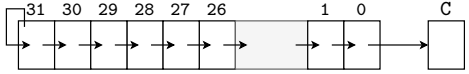
$1 \leq n \leq 32$

LSL
Logical Shift Left



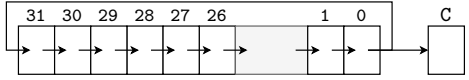
$0 \leq n \leq 31$

ASR
Arithmetic Shift Right



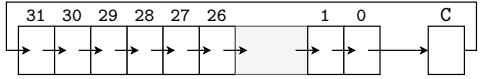
$1 \leq n \leq 32$

ROR
Rotate Shift Right



$1 \leq n \leq 32$

RRX
Rotate Right Extended



$n = 1$

Instrucción LSR

Desplazamiento lógico a la derecha (no signado).

```
LSR{S}{cond} Rd, Rm, Rs
LSR{S}{cond} Rd, Rm, #n
```

En donde:

- `Rd` → Registro donde se cargará el resultado.
- `Rm` → Registro que contiene el dato al que se le aplicará el desplazamiento.
- `Rs` → Registro que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).
- `#n` → Constante que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

```
Rd, Rm, Rs
Rd, Rm, #n
```

Instrucción LSL

Desplazamiento lógico a la izquierda (no signado).

```
LSL{S}{cond} Rd, Rm, Rs
LSL{S}{cond} Rd, Rm, #n
```

En donde:

- `Rd` → Registro donde se cargará el resultado.
- `Rm` → Registro que contiene el dato al que se le aplicará el desplazamiento.
- `Rs` → Registro que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 31$).
- `#n` → Constante que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 31$).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

```
Rd, Rm, Rs
Rd, Rm, #n
```

Instrucción ASR

Desplazamiento aritmético a la derecha (signado).

```
ASR{S}{cond} Rd, Rm, Rs
ASR{S}{cond} Rd, Rm, #n
```

En donde:

- `Rd` → Registro donde se cargará el resultado.
- `Rm` → Registro que contiene el dato al que se le aplicará el desplazamiento.
- `Rs` → Registro que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).
- `#n` → Constante que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

```
Rd, Rm, Rs
Rd, Rm, #n
```

Instrucción ROR

Desplazamiento de rotación a la izquierda.

```
ROR{S}{cond} Rd, Rm, Rs
ROR{S}{cond} Rd, Rm, #n
```

En donde:

- `Rd` → Registro donde se cargará el resultado.
- `Rm` → Registro que contiene el dato al que se le aplicará el desplazamiento.
- `Rs` → Registro que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).
- `#n` → Constante que indica la cantidad de posiciones de desplazamiento ($1 \leq n \leq 32$).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

```
Rd, Rm, Rs
Rd, Rm, #n
```

Instrucción RRX

Desplazamiento de rotación extendida a la derecha.

```
RRX{S}{cond} Rd, Rm
```

En donde:

- `Rd` → Registro donde se cargará el resultado.
- `Rm` → Registro que contiene el dato al que se le aplicará el desplazamiento (de un solo bit).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C.

Operandos:

`Rd, Rm`

Instrucción ADD

Suma aritmética.

```
ADD{S}{cond} {Rd,} Rn, <Op2>
ADD{S}{cond} {Rd,} Rn, #imm12
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Registro que contiene al primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).
- `#imm12` → Segundo operando (dato de 12 bits).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C, V.

Operandos:

```
{Rd,} Rn, <Op2>
{Rd,} Rn, #imm12
```

Instrucción SUB

Resta aritmética (sustracción).

```
SUB{S}{cond} {Rd,} Rn, <Op2>
SUB{S}{cond} {Rd,} Rn, #imm12
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Registro que contiene al primer operando (si se omite `Rd`, almacenará el resultado).
- `<Op2>` → Segundo operando (flexible).
- `#imm12` → Segundo operando (dato de 12 bits).

Sufijos opcionales:

- `S` → Actualización del registro de estados (banderas).
  → N, Z, C, V.

Operandos:

```
{Rd,} Rn, <Op2>
{Rd,} Rn, #imm12
```

Instrucción CMP

Comparación (sustracción).

```
CMP{cond} Rn, <Op2>
```

En donde:

- `Rn` → Registro que contiene al primer operando.
- `<Op2>` → Dato para comparar (operando flexible).

Descripción: Actualizar el registro de estados (banderas) al aplicar la sustracción de `Rn` menos `<Op2>`.

Características principales:

- No se almacena el resultado.
- Permite la toma de decisiones a partir de la actualización del registro de estados (banderas).

Instrucción MUL
Multiplicación.

```
MUL{S}{cond} {Rd,} Rn, Rm
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Registro que contiene al primer operando (si se omite `Rd`, almacenará el resultado).
- `Rm` → Registro que contiene al segundo operando (dato de 32 bits).

Descripción: Realizar la operación `Rn * Rm` y almacenar el resultado en `Rd` (o en `Rn`).

Características principales:

- Utiliza operandos de 32 bits y se obtiene un resultado también de 32 bits.
- Para el resultado se consideran únicamente los 32 bits menos significativos.
- Se puede realizar con números signados y no signados.
- El sufijo opcional `S` solo actualiza las banderas N, Z.

Instrucciones MLA y MLS

Multiplicación con acumulador (MLA) y con sustracción (MLS).

```
MLA{cond} Rd, Rn, Rm, Ra
MLS{cond} Rd, Rn, Rm, Ra
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Registro que contiene al primer operando (dato de 32 bits).
- `Rm` → Registro que contiene al segundo operando (dato de 32 bits).
- `Ra` → Registro que contiene al acumulador o sustracción (dato de 32 bits).

Descripción:

`MLA` → Realizar la operación `Ra` + (`Rn` * `Rm`) y almacenar el resultado en `Rd`.
`MLS` → Realizar la operación `Ra` - (`Rn` * `Rm`) y almacenar el resultado en `Rd`.

Instrucciones de multiplicación y división

Instrucciones UDIV y SDIV

División no signada (UDIV) y signada (SDIV).

```
UDIV{cond} {Rd,} Rn, Rm
SDIV{cond} {Rd,} Rn, Rm
```

En donde:

- `Rd` → Registro donde se cargará el resultado (opcional).
- `Rn` → Registro que contiene al primer operando (si se omite `Rd`, almacenará el resultado).
- `Rm` → Registro que contiene al segundo operando (dato de 32 bits).

Descripción:

`UDIV` → Realizar la división no signada de `Rn` / `Rm` y almacenar el resultado en `Rd` (o en `Rn`).
`SDIV` → Realizar la división signada de `Rn` / `Rm` y almacenar el resultado en `Rd` (o en `Rn`).

Características principales:

- Si `Rn` no es divisible por `Rm`, el resultado se redondea a cero.

Instrucciones B y BX

Salto (`B`) y salto indirecto (`BX`).

```
B{cond} label
BX{cond} Rm
```

En donde:

- `label` → Etiqueta que expresa una dirección relativa al PC.
- `Rm` → Registro que indica la dirección a saltar.

Descripción:

`B` → Salto incondicional (inmediato) a dirección del PC referenciada por `label`.
`BX` → Salto indirecto a la dirección del PC especificada por `Rm`.

Instrucciones BL y BLX

Salto con liga (`BL`) y salto indirecto con liga (`BLX`).

```
BL{cond} label
BLX{cond} Rm
```

En donde:

- `label` → Etiqueta que expresa una dirección relativa al PC.
- `Rm` → Registro que indica la dirección a saltar.

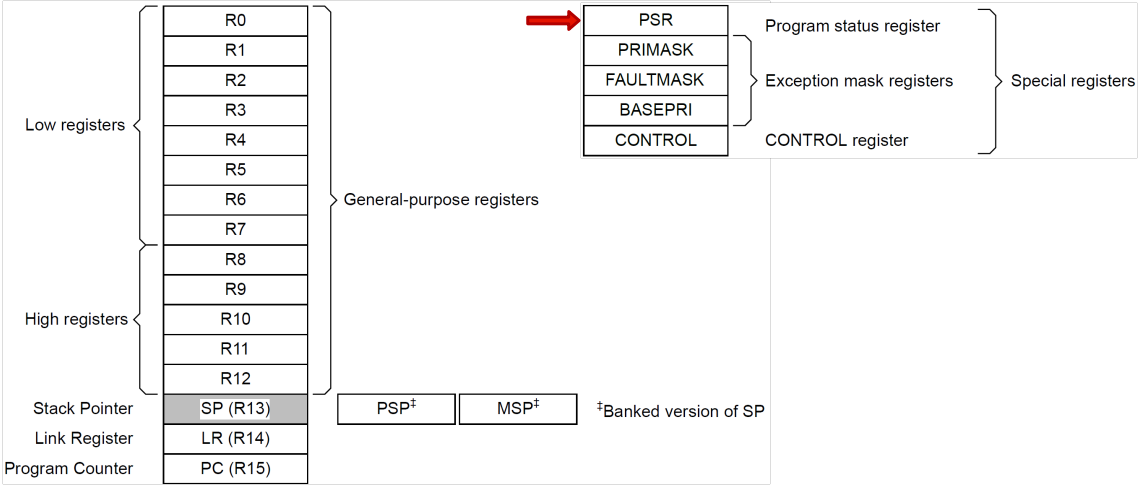
Descripción:

`BL` → Salto a la subrutina ubicada en la dirección del PC referenciada por `label`.
`BLX` → Salto indirecto a la subrutina ubicada en la dirección del PC especificada por `Rm`.

Características principales:

- `BL` y `BLX` escriben la dirección de la siguiente instrucción al registro de liga `LR` (`R14`).

Set de registros en el ARM Cortex-M4



Registro de estados (PSR)

Program Status Register (PSR)

Type RW, reset 0x0100.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
N	Z	C	V	Q	ICI / IT		THUMB	reserved				GE			
Type	RW	RW	RW	RW	RW	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICI / IT						reserved		ISRNUM							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

El registro de estados (PSR o xPSR) tiene tres funciones principales:

- Registro de estado de aplicación del programa (**APSR**) – bits 31:27, bits 19:16.
- Registro de estado de ejecución del programa (**EPSR**) – bits 26:24, bits 15:10.
- Registro de estado de interrupción del programa (**IPSR**) – bits 7:0.

Registro de estados (PSR)

Program Status Register (PSR)

Type RW, reset 0x0100.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
N	Z	C	V	Q	ICI / IT		THUMB	reserved				GE			
Type	RW	RW	RW	RW	RW	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICI / IT						reserved		ISRNUM							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Registro de estado de aplicación del programa (**APSR**) – bits 31:27, bits 19:16.

- Indica el estado actual de las banderas de condición para una instrucción ejecutada previamente.
  - Negativo/Negative – N (bit 31).
  - Cero/Zero – Z (bit 30).
  - Acarreo/Carry – C (bit 29).
  - Sobreflujo / Overflow – O (bit 28).

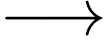
N – Negativo/Negative
Bandera que indica que el resultado previo calculado por el procesador entregó como resultado un dato negativo (con el bit más significativo en "1").

Por ejemplo, si el procesador ejecutará la instrucción:

```
ORRS R0, R1, R2
```

Si previamente se tenia:

```
R0 = 0x1234 5678
R1 = 0x6111 1111
R2 = 0x8000 0000
```



Los resultados serán:

```
R0 = 0xE111 1111
R1 = 0x6111 1111
R2 = 0x8000 0000
```

Dando como resultado que la bandera **N** se habilite ($N = 1$).

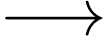
Z – Cero/Zero
Bandera que indica que el resultado previo calculado por el procesador entregó como resultado un cero.

Por ejemplo, si el procesador ejecutará la instrucción:

```
ANDS R0, R1, R2
```

Si previamente se tenia:

```
R0 = 0x1234 5678
R1 = 0xAAAA AAA5
R2 = 0x5555 5555
```



Los resultados serán:

```
R0 = 0x0000 0005
R1 = 0xAAAA AAAA
R2 = 0x5555 5555
```

Dando como resultado que la bandera **Z** se deshabilite ($Z = 0$).

C – Acarreo/Carry

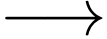
Bandera que indica que el resultado previo de una operación no signada es incorrecto, especificando que el resultado supera el valor máximo que puede ser representado con la longitud de palabra.

Por ejemplo, si el procesador ejecutará la instrucción:

```
ADDS R0, R1, R2
```

Si previamente se tenia:

```
R0 = 0x1234 5678
R1 = 0xFFFF FFFF
R2 = 0x0000 0001
```



Los resultados serán:

```
R0 = 0x0000 0000
R1 = 0xFFFF FFFF
R2 = 0x0000 0001
```

Dando como resultado que la bandera **C** se habilite ($C = 1$).

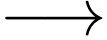
V – Sobreflujo/Overflow
Bandera que indica que el resultado previo de una operación signada (suma o resta) es incorrecto.

Por ejemplo, si el procesador ejecutará la instrucción:

```
ADDS R0, R1, R2
```

Si previamente se tenia:

```
R0 = 0x1234 5678
R1 = 0x5555 5555
R2 = 0x5555 5555
```



Los resultados serán:

```
R0 = 0xAAAA AAAA
R1 = 0x5555 5555
R2 = 0x5555 5555
```

Dando como resultado que la bandera **V** se habilite ($V = 1$), ya que la suma de dos datos positivos está entregando un resultado negativo.

Saltos condicionales

Saltos que se realizarán solo si se cumple con la condición específica.

Mnemónico	Operandos	Significado	Banderas de condición
BEQ	label o Rm	Igual	Z=1
BNE	label o Rm	No igual	Z=0
BCS	label o Rm	Mayor o igual, no signado $\geq$	C=1
BHS	label o Rm	Mayor o igual, no signado $\geq$	C=1
BCC	label o Rm	Menor, no signado $<$	C=0
BLO	label o Rm	Menor, no signado $<$	C=0
BMI	label o Rm	Negativo	N=1
BPL	label o Rm	Positivo o cero	N=0
BVS	label o Rm	Overflow	V=1
BVC	label o Rm	No overflow	V=0
BHI	label o Rm	Mayor, no signado $>$	C=1 y Z=0
BLS	label o Rm	Mayor o igual, no signado $\leq$	C=0 o Z=1
BGE	label o Rm	Mayor que o igual, signado $\geq$	N=V
BLT	label o Rm	Menor que, signado $<$	N!=V
BGT	label o Rm	Mayor que, signado $>$	Z=0 y N=V
BLE	label o Rm	Menor que o igual, signado $\leq$	Z=1 y N!=V

Instrucción IT

If-then.

$$\texttt{IT\{x\{y\{z\}\}}}\ \texttt{cond}$$

En donde:

- `x` → Especifica la condición de la segunda instrucción en el bloque `IT`.
- `y` → Especifica la condición de la tercera instrucción en el bloque `IT`.
- `z` → Especifica la condición de la cuarta instrucción en el bloque `IT`.
- `cond` → Especifica la condición de la primera instrucción en el bloque `IT`.

Condiciones:

`T` → `then`. Aplica la condición `cond` de la instrucción.
`E` → `else`. Aplica la condición `cond` inversa de la instrucción.

Instrucción NOP
No operación.

$$\texttt{NOP}\{\texttt{cond}\}$$

En donde:

- `cond` → Condición opcional.

Descripción: `NOP` representa nada, sin necesariamente consumir tiempo, ya que el procesador puede remover la instrucción del pipeline antes de que alcance la etapa de ejecución.

Características principales:

- No se altera el registro de estados (banderas).

Sufijos de código de condición

Una instrucción con código de condición se ejecutará solo si el registro de estados cumple con la condición específica. De este modo, se puede reducir el número de instrucciones de salto y control. Las instrucciones con código de condición requieren de la instrucción `IT` como instrucción previa.

Sufijo	**Banderas**	**Significado**
`EQ`	`Z=1`	Igual
`NE`	`Z=0`	No igual
`CS / HS`	`C=1`	Mayor o igual, no signado $\geq$
`CC / LO`	`C=0`	Menor, no signado $<$
`MI`	`N=1`	Negativo
`PL`	`N=0`	Positivo o cero
`VS`	`V=1`	Overflow
`VC`	`V=0`	No overflow
`HI`	`C=1 y Z=0`	Mayor, no signado $>$
`LS`	`C=0 o Z=1`	Mayor o igual, no signado $\leq$
`GE`	`N=V`	Mayor que o igual, signado $\geq$
`LT`	`N!=V`	Menor que, signado $<$
`GT`	`Z=0 y N=V`	Mayor que, signado $>$
`LE`	`Z=1 y N!=V`	Menor que o igual, signado $\leq$
`AL`	`Cualquier valor`	Siempre. Condición por defecto cuando no se especifica un sufijo.

Tabla de instrucciones

Clasificación	Instrucción	Descripción	Banderas
Acceso a memoria	`LDR`	Cargar datos desde la memoria a un registro.	-
	`STR`	Cargar datos desde un registro a memoria.	-
Generales de procesamiento de datos	`MOV`	Cargar el contenido de un lugar a otro.	N, Z, C
	`MOVW / MOVT`	Cargar el contenido de un lugar a otro (16 bits).	-
	`AND`	AND lógico de 32 bits.	N, Z, C
	`ORR`	OR lógico de 32 bits.	N, Z, C
	`ORN`	NOT OR lógico de 32 bits.	N, Z, C
	`EOR`	XOR lógico de 32 bits.	N, Z, C
	`BIC`	Bit clear.	N, Z, C
	`LSR / LSL`	Desplazamiento lógico a la derecha / izquierda (no signado).	N, Z, C
	`ASR`	Desplazamiento aritmético a la derecha (signado).	N, Z, C
	`ROR`	Desplazamiento de rotación a la izquierda.	N, Z, C
	`RXX`	Desplazamiento de rotación extendida a la derecha.	N, Z, C
	`ADD`	Suma aritmética.	N, Z, C, V
	`SUB`	Resta aritmética (sustracción).	N, Z, C, V
	`CMP`	Comparación (sustracción).	N, Z, C, V
Multiplicación y división	`MUL`	Multiplicación con resultado de 32 bits.	N, Z
	`MLA / MLS`	Multiplicación con acumulador / sustracción.	N, Z (MLA) / - (MLS)
	`UDIV / SDIV`	División no signada / signada.	-
Salto y control	`B / BX`	Salto incondicional / indirecto incondicional.	-
	`BL / BLX`	Salto con liga / indirecto con liga.	-
	`Bxx`	Saltos condicionales.	-
	`IT`	If-then.	-
Varias	`NOP`	No operación.	-

Tarea 3 – Código básico en lenguaje ensamblador

Desarrollar un código en ensamblador que realice lo siguiente:

1. OR lógico de `0xAA22 1177` con `0x8136 2469`
2. XOR lógico del resultado con `0xF0F0 F0F0`
3. Limpiar los 8 bits menos significativos del resultado
4. Aplicar un desplazamiento lógico hacia la derecha de 16 bits
5. Comparar el resultado con `23,500` e indicar si es mayor, menor o igual
6. Guardar el último resultado calculado en la localidad de memoria `0x2000 0000`

Para la entrega del reporte:

- El código se debe de comentar.
- Se deben presentar los resultados que demuestren los puntos *5.* y *6.*