Universidad Nacional Autónoma de México
Facultad de Ingeniería

Microprocesadores y Microcontroladores

Tema 5

Programación estructurada en lenguaje ensamblador

M. I. Christo Aldair Lara Tenorio

2025-1

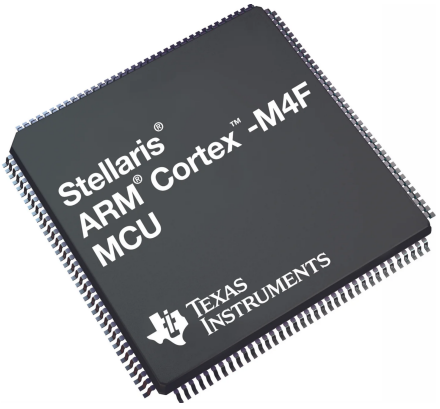
Tabla de contenidos

Objetivo del tema

Objetivo general:
El alumno diseñará programas de aplicación en lenguaje ensamblador.

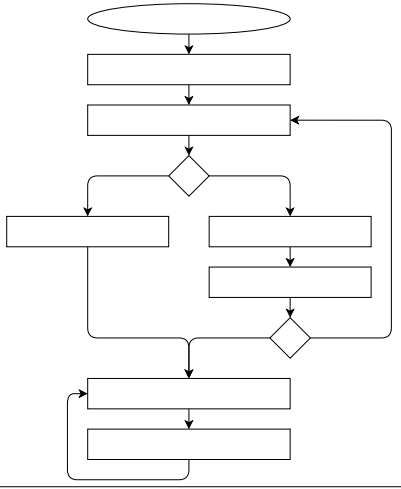
Contenido:

5.1. Herramientas de diseño y documentación.
5.2. Construcción de estructuras de control.
5.3. Almacenamiento de datos.
5.4. Estructura de un programa.
5.5. Pase de parámetros.



Programación estructurada

Paradigma de programación que utiliza estructuras de control para mejorar la claridad y eficiencia de un programa.



Características principales:

- Control del flujo del programa.
- Toma de decisiones.
- Modularidad.
- Procesos iterativos.
- Legibilidad.
- Eficiencia en memoria.

En lenguaje ensamblador:

- No existen estructuras de control predefinidas.
- Se pueden implementar utilizando instrucciones de salto y etiquetas.

Código 02: Saltos incondicionales

```
01          .global main            ; Declaración del archivo con la función principal (main)
02
03  counter .equ 1                  ; Declaración de la constante counter = 1
04
05  main:                           ; Inicio del código principal
06
07          MOV  R0, #0             ; R0 = 0
08
09  loop                            ; Etiqueta loop
10          ADD  R0, #counter       ; R0 = R0 + counter
11          B    loop               ; Saltar a loop
12
13  end     B    end
14          .end                    ; Final del archivo de código fuente
```

Saltos condicionales

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

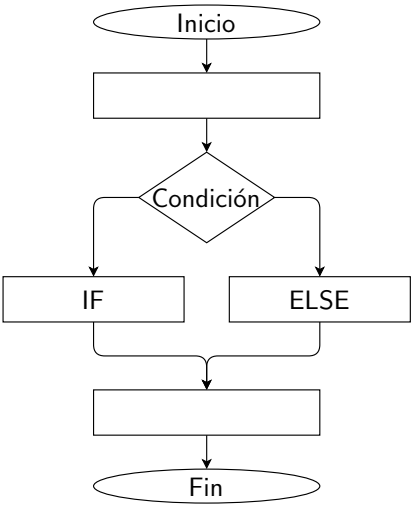
Mnemónico	Operandos	Significado	Banderas de condición
BEQ	label o Rm	Igual	Z=1
BNE	label o Rm	No igual	Z=0
BCS	label o Rm	Mayor o igual, no signado $\geq$	C=1
BHS	label o Rm	Mayor o igual, no signado $\geq$	C=1
BCC	label o Rm	Menor, no signado $<$	C=0
BLO	label o Rm	Menor, no signado $<$	C=0
BMI	label 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

Código 03: Saltos condicionales

```
01          .global main            ; Declaración del archivo con la función principal (main)
02
03  counter .equ 1                  ; Declaración de la constante counter = 1
04  lap     .equ 5                  ; Declaración de la constante lap = 5
05
06  main:                           ; Inicio del código principal
07
08          MOV  R1, #0             ; R1 = 0
09  loop2   MOV  R0, #0             ; R0 = 0
10
11  loop                            ; Etiqueta loop
12          ADD  R0, #counter       ; R0 = R0 + counter
13          CMP  R0, #lap           ; ¿R0 = #lap?
14          BNE  loop               ; Si Z=0 => saltar a loop
15          ADD  R1, #1             ; R1 = R1 + 1
16          B    loop2              ; Saltar a loop2
17
18  end     B    end
19          .end                    ; Final del archivo de código fuente
```

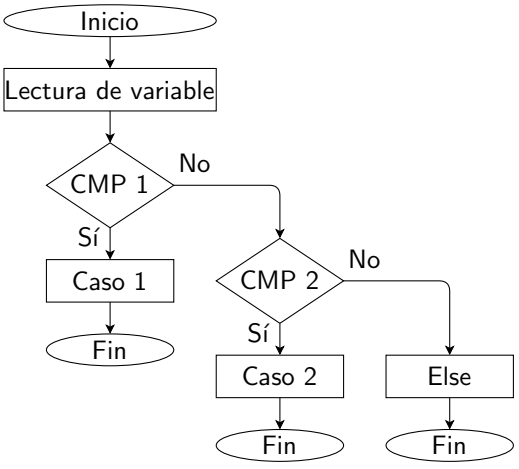
Estructuras de control: IF - ELSE (código 04)

```
01          .global main
02
03  main:
04
05          MOV  R0, #0
06          MOV  R1, #5
07
08  inicio  CMP  R1, #5      ; Condición
09          BNE  else
10
11  if      ADD  R0, #1      ; Código del IF
12          ADD  R0, #1
13          B    fin
14
15  else    ADD  R0, #2      ; Código del ELSE
16          ADD  R0, #2
17
18  fin     MOV  R2, R0      ; Fin del IF-ELSE
19
20  end     B    end
21          .end
```



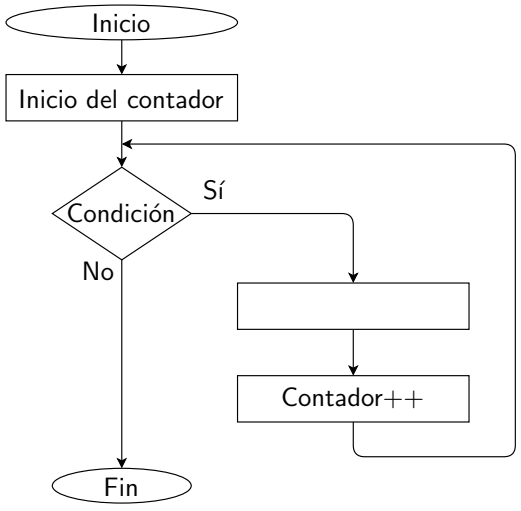
Estructuras de control: SWITCH-CASE (código 05)

```
01          .global main
02
03  main:
04
05          MOV  R0, #0
06          MOV  R1, #2
07
08  inicio  CMP  R1, #1      ; Condición 1
09          BEQ  caso1
10          CMP  R1, #2      ; Condición 2
11          BEQ  caso2
12          B    else        ; Condición ELSE
13
14  caso1   ADD  R0, #1      ; Código del caso1
15          B    fin
16
17  caso2   ADD  R0, #2      ; Código del caso2
18          B    fin
19
20  else    ADD  R0, #3      ; Código del ELSE
21
22  fin     MOV  R2, R0      ; Fin del SWITCH-CASE
23
24  end     B    end
25          .end
```



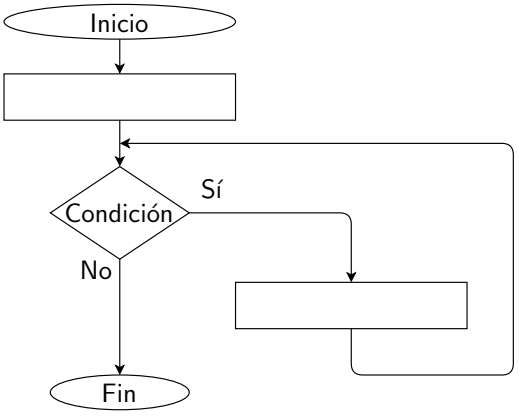
Estructuras de control: FOR (código 06)

```
01          .global main
02
03  main:
04
05          MOV  R0, #0
06          MOV  R1, #0      ; Inicio del contador
07
08  for     CMP  R1, #5      ; Condición
09          BEQ  fin
10
11          ADD  R0, #10     ; Código del FOR
12          ADD  R0, #10
13
14          ADD  R1, #1      ; Contador++
15          B    for
16
17  fin     MOV  R2, R0      ; Fin del FOR
18
19  end     B    end
20          .end
```



Estructuras de control: WHILE (código 07)

```
01          .global main
02
03  main:
04
05          MOV  R0, #0
06          MOV  R1, #5      ; Variable de control
07
08  while   CMP  R1, #5      ; Condición
09          BNE  fin
10
11          ADD  R0, #10     ; Código del WHILE
12          ADD  R0, #10
13
14          B    while
15
16  fin     MOV  R2, R0      ; Fin del WHILE
17
18  end     B    end
19          .end
```



Ejecución condicional de instrucciones

- La mayoría de las instrucciones de procesamiento de datos pueden actualizar (de manera opcional) las banderas del registro de estado (*SPR*).
- A partir del valor de las banderas de estado se puede condicionar la ejecución de una instrucción.

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

- El sufijo opcional de condición permite que el procesador pruebe una condición basada en las banderas de estados. Si la prueba de condición falla, la instrucción:
  - No se ejecuta.
  - No escribe datos en su registro destino.
  - No afecta el valor de las banderas de estado.
  - No genera ninguna excepción.
- Las instrucciones condicionales (excepto por los saltos condicionales) deben estar dentro de un bloque de instrucción `IT`.

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.

Instrucción IT (código 08)
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.

Cada instrucción dentro del bloque `IT` debe especificar un sufijo de código condicional, que puede seguir la misma lógica o la inversa.

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

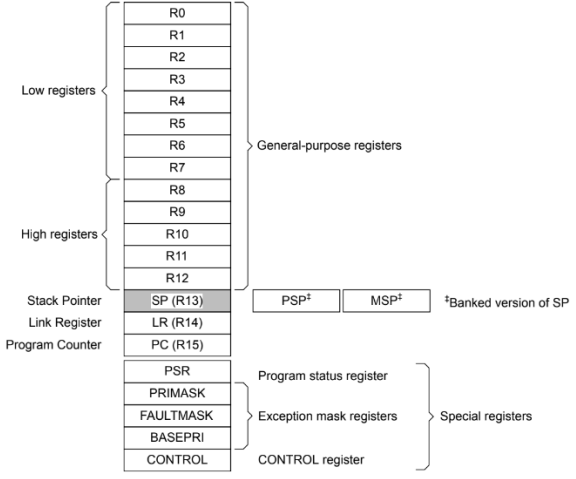
Registro de estado de ejecución del programa (**EPSR**) – bits 26:24, bits 15:10.

- Campo de Instrucción Interrumpible-Continuable (ICI).
- Bit de estado Thumb.
- Campo de estado de ejecución para la instrucción `IT`.

Almacenamiento de datos

Forma en que se guardan los datos.

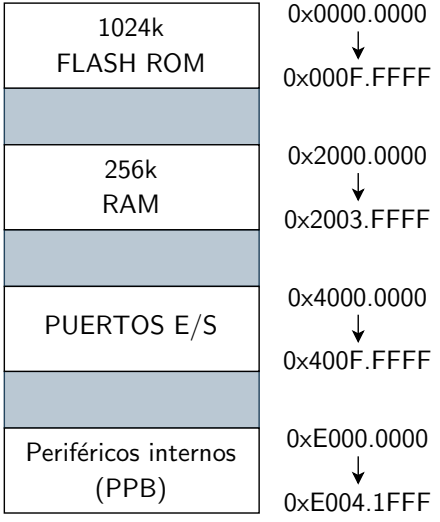
- **Registros.**
  Elementos de almacenamiento que permiten un acceso rápido a los datos.



Almacenamiento de datos

Forma en que se guardan los datos.

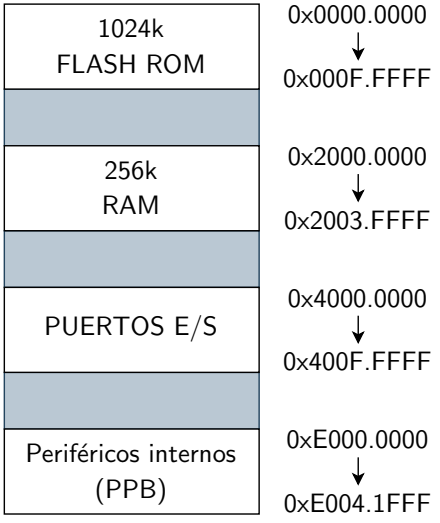
- **Registros.**
  Elementos de almacenamiento que permiten un acceso rápido a los datos.
- **Memoria (segmentos de memoria).**
  Áreas definidas de la memoria con propósitos específicos.



Almacenamiento de datos

Forma en que se guardan los datos.

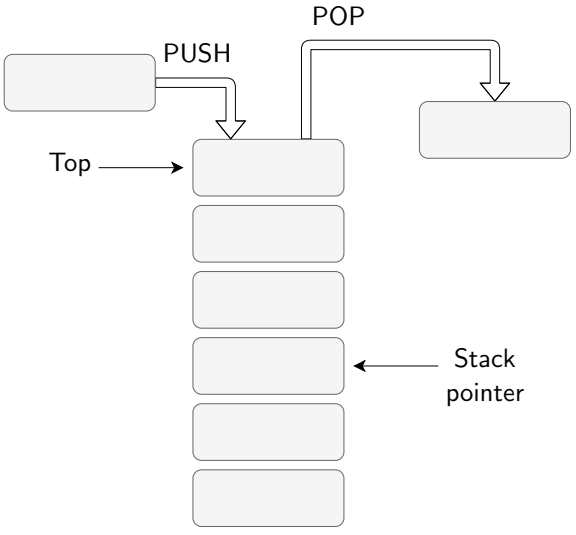
- **Registros.**
  Elementos de almacenamiento que permiten un acceso rápido a los datos.
- **Memoria (segmentos de memoria).**
  Áreas definidas de la memoria con propósitos específicos.
- **Variables y constantes.**
  Asignación de datos en zonas de memoria reservadas.



Almacenamiento de datos

Forma en que se guardan los datos.

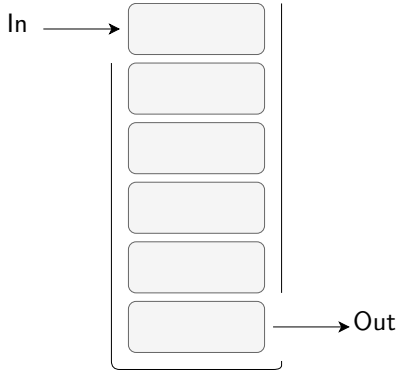
- **Registros.**
  Elementos de almacenamiento que permiten un acceso rápido a los datos.
- **Memoria (segmentos de memoria).**
  Áreas definidas de la memoria con propósitos específicos.
- **Variables y constantes.**
  Asignación de datos en zonas de memoria reservadas.
- **Pila (stack).**
  Estructura de datos de tipo LIFO que permite almacenar datos de forma temporal.



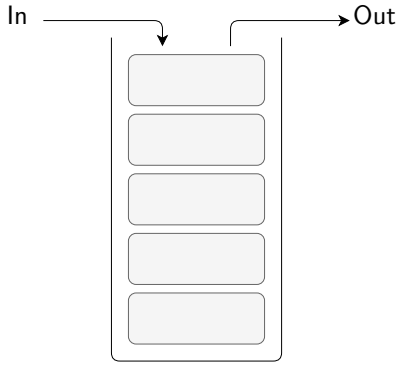
FIFO y LIFO

Tipos de estructuras de datos que permiten el manejo de estos de distintas formas.

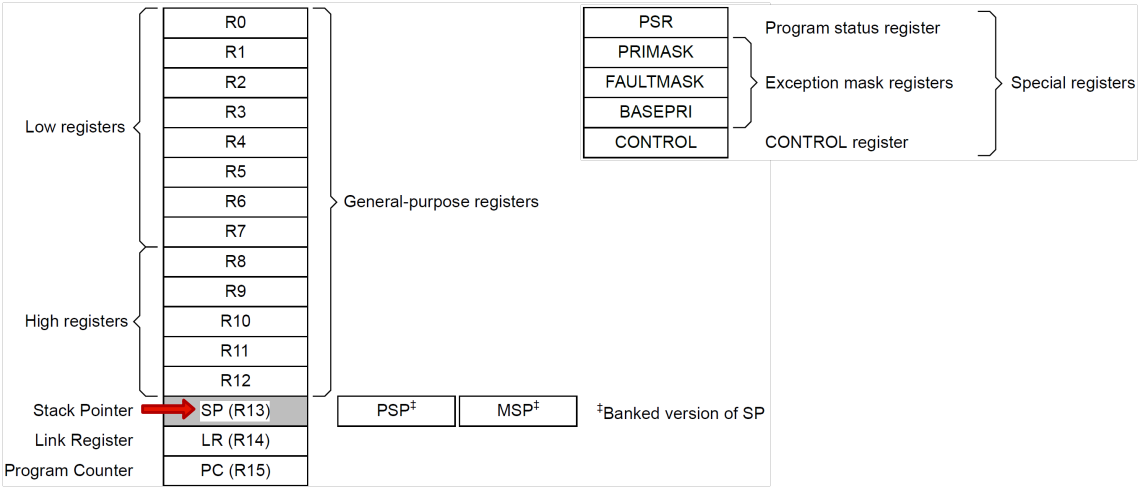
FIFO
First In, First Out.



LIFO
Last In, First Out.



Set de registros en el ARM Cortex-M4



Instrucciones PUSH y POP (código 09)

Instrucciones de acceso a memoria que permiten escribir (PUSH) y leer (pop) del stack, reduciendo la cantidad de registros utilizados para almacenar datos.

```
PUSH{cond} reglist
POP{cond} reglist
```

En donde:

- `reglist` → Lista de registros, encerrados por { }.

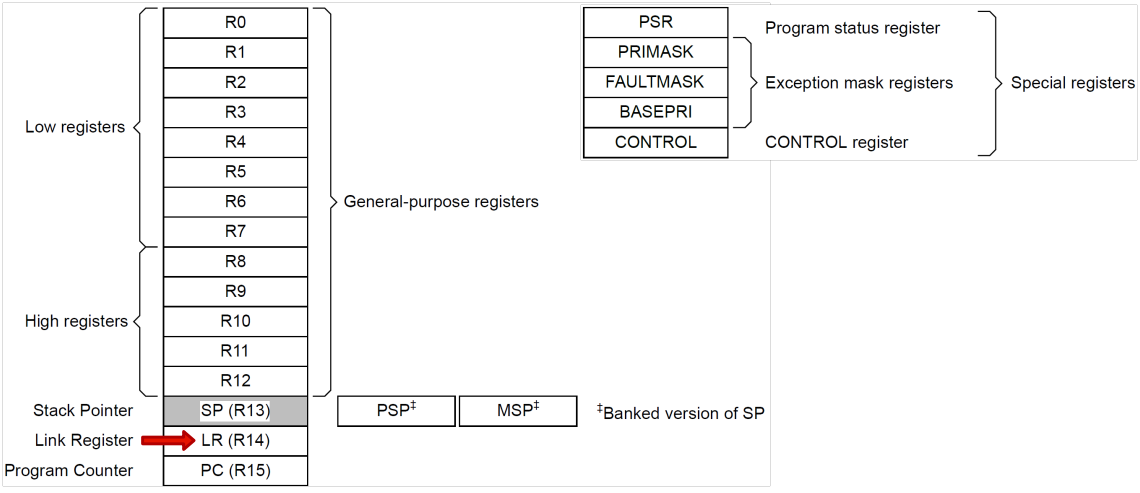
Descripción:

- `PUSH` → Almacena registros en el stack, almacenando el registro con el número más alto en la dirección de memoria más alta.
- `POP` → Carga registros desde el stack, cargando en el registro con el número más alto el dato de la dirección de memoria más alta.

Estructura básica de un programa en lenguaje ensamblador

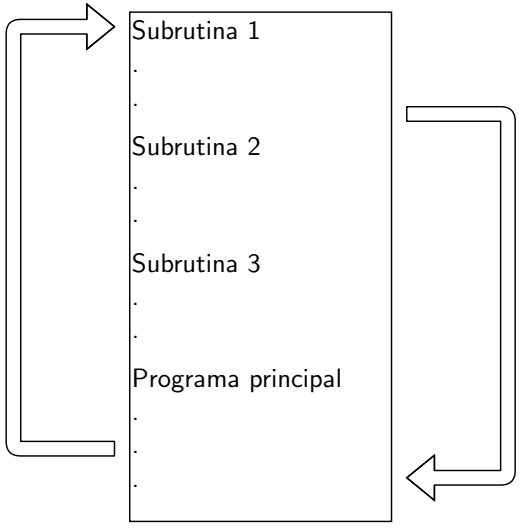
```
01          .global main          ; Declaración del archivo con la función principal (main)
02
03          .data                 ; Sección para escribir en memoria de datos
04
05          .text                 ; Sección para escribir el código ejecutable (instrucciones)
06
07  POINTER .field 0x20000000, 32
08  DOS     .equ 0x02
09
10  loop    ADD R1, R0            ; Sección de etiquetas (subrutinas)
11          B   loop
12
13  main:                         ; Inicio del código principal
14          MOV  R0, #0x12
15          MOV  R1, #0x01
16
17          .end                  ; Final del archivo de código fuente
```

Set de registros en el ARM Cortex-M4



Subrutinas (código 10)

- Bloque de código que puede ser llamado desde diferentes partes de un programa principal.
- Contiene un segmento de código que realiza una función específica.
- Para definir una subrutina se utilizan las etiquetas.
- La dirección de retorno se almacena en el registro de liga `LR` (`R14`).
- En procesadores ARM se utiliza la instrucción `BL` para llamar a una subrutina y la instrucción `BX` para retornar al código principal.
- La dirección de retorno es la localidad de memoria que almacena la instrucción inmediata después de la instrucción `BL`.



Instrucciones BL, BX

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

```
BL{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:

`BL` → Salto a la subrutina ubicada en la dirección del PC referenciada por `label`.
`BX` → Salto indirecto a 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`).

Registros en un ARM Cortex-M4

Registros de propósito general (`R0 - R12`)

- Registros que pueden ser utilizados para almacenar datos temporales, direcciones de memoria o resultados de las operaciones realizadas en un programa.
- No están reservados para cumplir con un uso específico.

Apuntador del stack `SP` (`R13`)

- Registro que apunta al último dato almacenado en la pila (stack).

Registro de liga `RL` (`R14`)

- Registro que almacena la dirección de retorno de una subrutina.

Contador de programa `PC` (`R15`)

- Registro que almacena la dirección de memoria de la siguiente instrucción que se va a ejecutar, determinando el flujo del programa.

Registros de estado del programa `PSR`

- Registro que contiene a los bits de estado (banderas) y a otros bits que ayudan a determinar el estado del procesador.

Tarea 4 – Programa en ensamblador

A partir de la dirección `0x2000.0000` de la memoria RAM del microcontrolador están almacenados 20 valores de 1 byte cada uno (los valores se llenarán manualmente antes de ejecutar el programa). Los primeros 10 valores se denominan X_i y los siguientes 10 valores se denominan Y_i, en donde i va de 0 a 9.

Desarrollar un programa en lenguaje ensamblador que, a partir de los valores X_i y Y_i, calcule:

- $\sum_0^9 X_i$
- $\sum_0^9 Y_i$
- $\sum_0^9 X_i Y_i$
- $\sum_0^9 X_i^2$
- $\overline{X_i}$ (promedio de X_i).
- $\overline{Y_i}$ (promedio de Y_i).

Los resultados se almacenarán en formato de 32 bits a partir de la dirección `0x2000.0020`.

Además, los resultados que sean mayores a `0x28` deben almacenarse en el stack en el orden en el que fueron calculados.