
Flujo de Ejecución de un programa

Organización de las Computadoras - Franco Garcino Ruiz - 2020

Flujo de ejecución en Q

El flujo de ejecución de nuestra máquina Q (sin importar la versión) será secuencial, es decir, se ejecutarán las instrucciones en el orden en que se lean. Este flujo podrá desviarse solo en dos casos:

- a. Cuando se ejecute la instrucción *CALL* o *RET*
- b. Cuando se ejecute algún salto (condicional o incondicional)

En esta presentación se mostrarán ejemplos de todos los casos

Flujo sin desvíos

Un flujo sin desvíos en la ejecución ejecutará una instrucción tras otras en el orden que se presenten.

Ejemplo: El siguiente código está ensamblado desde 0x0000, y PC contiene la dirección 0x0000

```
MOV R0, 0x000A
```

```
ADD R0, 0x0006
```

```
RET
```

Tras la primera instrucción, vemos los siguientes cambios:

- PC se actualiza a 0002 (cambio secuencial)
- IR contiene la primer instrucción
- R0 actualiza su contenido a 000A

The screenshot shows the Qsim debugger interface. On the left, the 'Registros especiales' (Special Registers) section shows PC at 0002, SP at FFEF, and IR at 1800 000A. The 'Flags' section shows N=0, V=0, Z=0, and C=0. The 'Registros' (Registers) section shows R0 at 000A, R1 at 0000, R2 at 0000, R3 at 0000, R4 at 0000, R5 at 0000, R6 at 0000, and R7 at 0000. A log window at the bottom left shows the following messages:

```
[28/4 17:25] *****INFORMACION*****  
[28/4 17:25] El programa compilado ha sido cargado en la memoria con éxito  
[28/4 17:25] La instrucion actual ocupa: 2  
[28/4 17:25] Se decodifico la instrucion : MOV R0, 0x000A  
[28/4 17:25] Se guardado el resultado 000A en R0
```

The main memory window shows a table of memory addresses from 0000 to 0170. The address 0000 contains the value 1800, and address 0001 contains 000A. The rest of the memory is filled with 0000. The status bar at the bottom right shows the time 17:25 and date 28/5/2020.

Tras la siguiente instrucción, vemos los siguientes cambios:

- PC se actualiza a 0004 (cambio secuencial)
- IR contiene la segunda instrucción
- R0 actualiza su contenido a 0010

Qsim

Registros especiales

PC: 0004

SP: FFEF

IR: 2800 0006

Flags

N: 0, V: 0, Z: 0, C: 0

Registros

R0: 0010, R1: 0000, R2: 0000, R3: 0000, R4: 0000, R5: 0000, R6: 0000, R7: 0000

Desde: 0000 Hasta: 0200 Actualizar Inicio_0000_01F0_Fin

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	1900	000A	2800	0006	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00C0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0120	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0130	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0140	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0150	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0160	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0170	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Log:

[28/4 17:25] *****INFORMACION*****

[28/4 17:25] El programa compilado ha sido cargado en la memoria con éxito

[28/4 17:25] La instrucción actual ocupa: 2

[28/4 17:25] Se decodificó la instrucción: MOV R0, 0x000A

[28/4 17:25] Se guardado el resultado 000A en R0

[28/4 17:26] La instrucción actual ocupa: 2

[28/4 17:26] Se decodificó la instrucción: ADD R0, 0x0006

[28/4 17:26] Se guardado el resultado 0010 en R0

Ejecucion Paso a Paso: Buscar, Decodificar, Ejecutar

Ejecucion Completa: Busc-Decod-Ejecu, Edicion: Editar

17:26 28/5/2020

¿Cómo es el flujo de ejecución en este ejemplo?

El flujo de ejecución de este programa es:

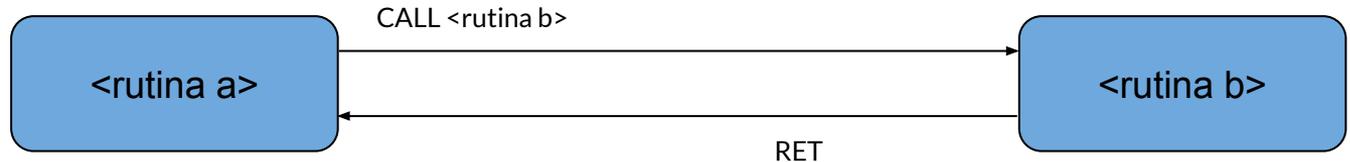
MOV R0, 0x000A PC: 0000 → 0002

ADD R0, 0x0006 PC: 0002 → 0004

Es decir, el flujo se ejecutó de manera secuencial.

Flujo con CALL y RET

Cuando se ejecuta una instrucción CALL lo que se está haciendo es **desviar** el flujo de ejecución de la rutina principal hacia otra ensamblada en la memoria, ejecutarla, y volver para seguir ejecutando la rutina principal.



PC 2 Y PC 3

Junto con las instrucciones CALL y RET, Q3 añade entre otras cosas dos estados nuevos de PC en la simulación de un programa (Sección 7.3 del libro):

- a. PC 2: es el valor que toma PC para ejecutar la siguiente instrucción en orden secuencial
 - b. PC 3: es el valor que toma PC tras la ejecución de la instrucción, siendo este valor la próxima celda a leer
-

Ejemplo

Supongamos entonces que se quiere calcular el triple de un valor, convenientemente se tiene una rutina que calcula el triple del valor almacenado en R0. Se arma entonces el siguiente código y datos:

- a. PC contiene 0000
 - b. La rutina principal está ensamblada desde 0x0000
 - c. La rutina *tripleSinMul* está ensamblada desde 0x0010
-

Ejemplo

```
MOV R0, 0x001D  
CALL tripleSinMul  
RET
```

```
tripleSinMul  ADD R0, R0  
              ADD R0, R0  
              RET
```

En esta imagen se puede observar el programa ensamblado y cargado en memoria.

Nuestra rutina principal se encuentra ensamblada desde la celda 0x0000 hasta la celda 0x0004.

La rutina *tripleSinMul* se encuentra ensamblada desde la celda 0x0010 hasta la celda 0x0012.

PC posee el valor 0000.

The screenshot shows the EquiSim debugger interface. The main window displays assembly code with addresses from 0000 to 0170. The PC register is set to 0000. The SP register is set to FFEF. The flags N, Z, V, and C are all set to 0. The registers R0 through R7 are all set to 0000. A message box indicates that the program has loaded successfully.

Registros especiales

PC: 0000

SP: FFEF

IR:

Flags

N: 0, V: 0

Z: 0, C: 0

Registros

R0: 0000, R1: 0000, R2: 0000, R3: 0000

R4: 0000, R5: 0000, R6: 0000, R7: 0000

[28/4 17:56] *****INFORMACION*****
[28/4 17:56] El programa compilado ha sido cargado en la memoria con éxito

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	1800	001D	B000	0010	C000	19E6										
0010	2820	2820	C000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00C0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0120	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0130	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0140	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0150	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0160	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0170	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Ejecucion Paso a Paso: Buscar, Decodificar, Ejecutar

Ejecucion Completa: Busc-Decod-Ejecu

Tras la ejecución de la primer instrucción vemos los siguientes cambios:

- PC posee el valor 0002 (cambio secuencial)
- IR se actualiza su contenido
- R0 posee el valor 001D

The screenshot shows the QSim debugger interface. On the left, the 'Registros especiales' (Special Registers) section displays: PC: 0002, SP: FFEF, IR: 1800 001D. Below this, the 'Flags' section shows N: 0, Z: 0, V: 0, and C: 0. The 'Registros' (Registers) section shows R0: 001D, R1: 0000, R2: 0000, R3: 0000, R4: 0000, R5: 0000, R6: 0000, and R7: 0000. A message window at the bottom left contains the following text: [28/4 17:56] *****INFORMACION***** [28/4 17:56] El programa compilado ha sido cargado en la memoria con exito [28/4 17:56] La intruccion actual ocupa: 2 [28/4 17:56] Se decodifico la instruccion : MOV R0, 0x001D [28/4 17:56] Se guardado el resultado 001D en R0. The main memory window shows a table of memory addresses from 0000 to 0170. The address 0000 contains the value 1800 001D, which is highlighted in green. The address 0001 contains 2820 2820 C000. All other memory addresses contain 0000. The status bar at the bottom right shows the time 17:56 and date 28/5/2020.

¿Cómo es el flujo de ejecución en este ejemplo?

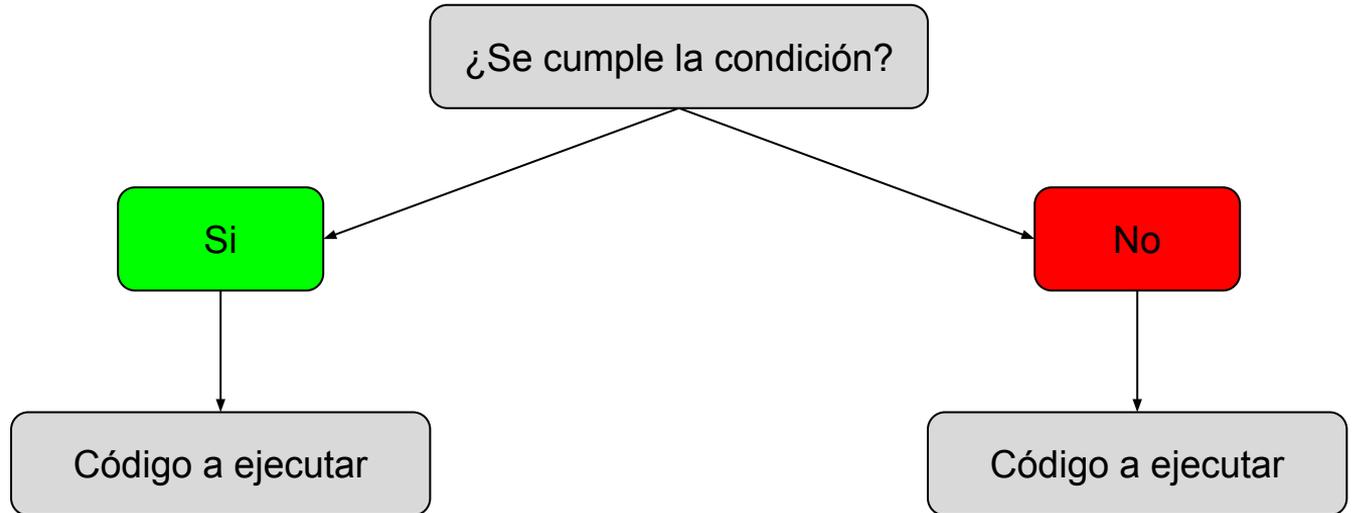
El flujo de ejecución de este programa es secuencial hasta que se ejecuta la instrucción CALL. Tras eso, se ejecutan las instrucciones de la <rutina b> (también de forma secuencial). Una vez terminada esta rutina, se regresa a la rutina que invocó a <rutina b>, y se siguen ejecutando sus instrucciones secuencialmente.

Flujo de la estructura condicional

Q4 añade la posibilidad de utilizar una estructura condicional para ejecutar una parte de código si se cumple una condición dada, caso contrario, se ejecutará otro fragmento de código. Para eso se añaden las siguientes instrucciones:

- a. CMP: permite comparar dos valores
 - b. Salto incondicional o absoluto: se ejecuta **siempre** (JMP)
 - c. Saltos condicionales o relativos: se ejecutan al cumplirse cierta condición (JGE, JNE, etc...)
-

Estructura condicional - Funcionamiento



Ejemplo

Se quiere saber si un alumno aprobó una materia de la carrera (se considera aprobado a un alumno con una nota **igual o mayor** a 40 puntos, básicamente, un 4). Para eso se arma el siguiente programa, el cual posee la siguiente documentación:

Requiere: en R6 un número en el rango [0..100] en BSS

Modifica: ---

Retorna: En R0 un 1 si aprobó, y 0 en caso contrario

Ejemplo

En este caso, el alumno en cuestión sacó una nota de 100/100, por ende, el valor que le vamos a pasar a R6 es 0064

```
MOV R6, 0x0064 // parametrización
```

```
    analisisDeNota CMP R6 0x0028
```

```
                        JGE materiaAprobada
```

```
                        MOV R0, 0x0000
```

```
                        JMP fin
```

```
materiaAprobada MOV R0, 0x0001
```

```
    fin RET
```

En la imagen se puede ver el programa cargado en memoria desde la celda 0x0000 y el valor con el cual empieza PC.

The screenshot shows the Qsim debugger interface. On the left, the 'Registros especiales' (Special Registers) section displays: PC: 0000, SP: FFFF, IR: (empty). Below this, the 'Flags' section shows N: 0, V: 0, Z: 0, and C: 0. The 'Registros' (Registers) section shows R0: 0074, R1: 0000, R2: 0000, R3: 0000, R4: 0000, R5: 0000, R6: 0000, and R7: 0000. A message box at the bottom left of the debugger area contains the text: [28/4 19:2] *****INFORMACION***** [28/4 19:2] El programa compilado ha sido cargado en la memoria con éxito.

At the top right, the memory range is set to 'Desde: 0000 Hasta: 0200' with an 'Actualizar' button and a link 'Inicio_0000_01F0_Fin'. The main memory window displays a table of memory addresses and their values:

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	1980	0064	6980	0028	FB04	1800	0000	A000	000B	1800	0001	C000	0000	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00C0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0120	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0130	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0140	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0150	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0160	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0170	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

At the bottom, the 'Ejecucion Paso a Paso' (Step-by-Step Execution) section contains buttons for 'Buscar' (Search), 'Decodificar' (Decode), 'Ejecutar' (Execute), and 'Busc-Decod-Ejecu' (Search-Decode-Execute).

Tras la ejecución de la primer instrucción vemos los siguientes cambios:

- PC posee el valor 0002 (cambio secuencial)
- IR se actualiza su contenido
- R6 ahora contiene el valor 0064

The screenshot shows the Qsim debugger interface. On the left, the 'Registros especiales' (Special Registers) section displays: PC: 0002, SP: FFFF, IR: 1980 0064. Below this, the 'Flags' section shows N: 0, V: 0, Z: 0, and C: 0. The 'Registros' (Registers) section shows R0: 0074, R1: 0000, R2: 0000, R3: 0000, R4: 0000, R5: 0000, R6: 0064, and R7: 0000. A log window at the bottom left shows the following messages: [28/4 19:2] *****INFORMACION*****, [28/4 19:2] El programa compilado ha sido cargado en la memoria con éxito, [28/4 19:3] La instrucción actual ocupa: 2, [28/4 19:3] Se decodificó la instrucción : MOV R6, 0x0064, [28/4 19:3] Se guardado el resultado 0064 en R6. The main window displays a memory dump from address 0000 to 0170. The dump shows hexadecimal values for each byte, with the first few bytes being 1980, 0064, 6980, 0028, FB04, 1800, 0000, A000, 000B, 1800, 0001, C000. The rest of the memory contains 0000. The interface includes buttons for 'Ejecucion Paso a Paso', 'Ejecucion Completa', and 'Edicion', along with 'Buscar', 'Decodificar', 'Ejecutar', 'Busc-Decod-Ejecu', and 'Editar'.

¿Que hubiese ocurrido en caso de no cumplirse la condición?

En caso de no cumplirse la condición, se habrían seguido ejecutado las instrucciones de manera secuencial, las cuales representaban la porción de código asignada para cuando la condición no se cumpliera.

¿Cómo es el flujo de ejecución en este ejemplo?

El flujo de ejecución de este programa es secuencial hasta el momento en que se ejecuta el salto condicional. Como se cumple la condición, se desvía el flujo hacia la porción de código que debe ejecutarse