Técnicas Digitales II

Bifurcaciones

Dr.Ing.Steiner Guillermo

Bifurcación

- La ejecución condicional, es solo una vía para ejecutar una instrucción o grupo de ellas en función de una condición.
- Esto es útil para manejo de una lista pequeña de instrucciones condicionadas.
- Se complica para grandes cantidades de código condicionados y no permite que un código se repita.
- Para estas aplicaciones, ARM como otras arquitecturas utiliza la instrucciones de bifurcación *branch.*

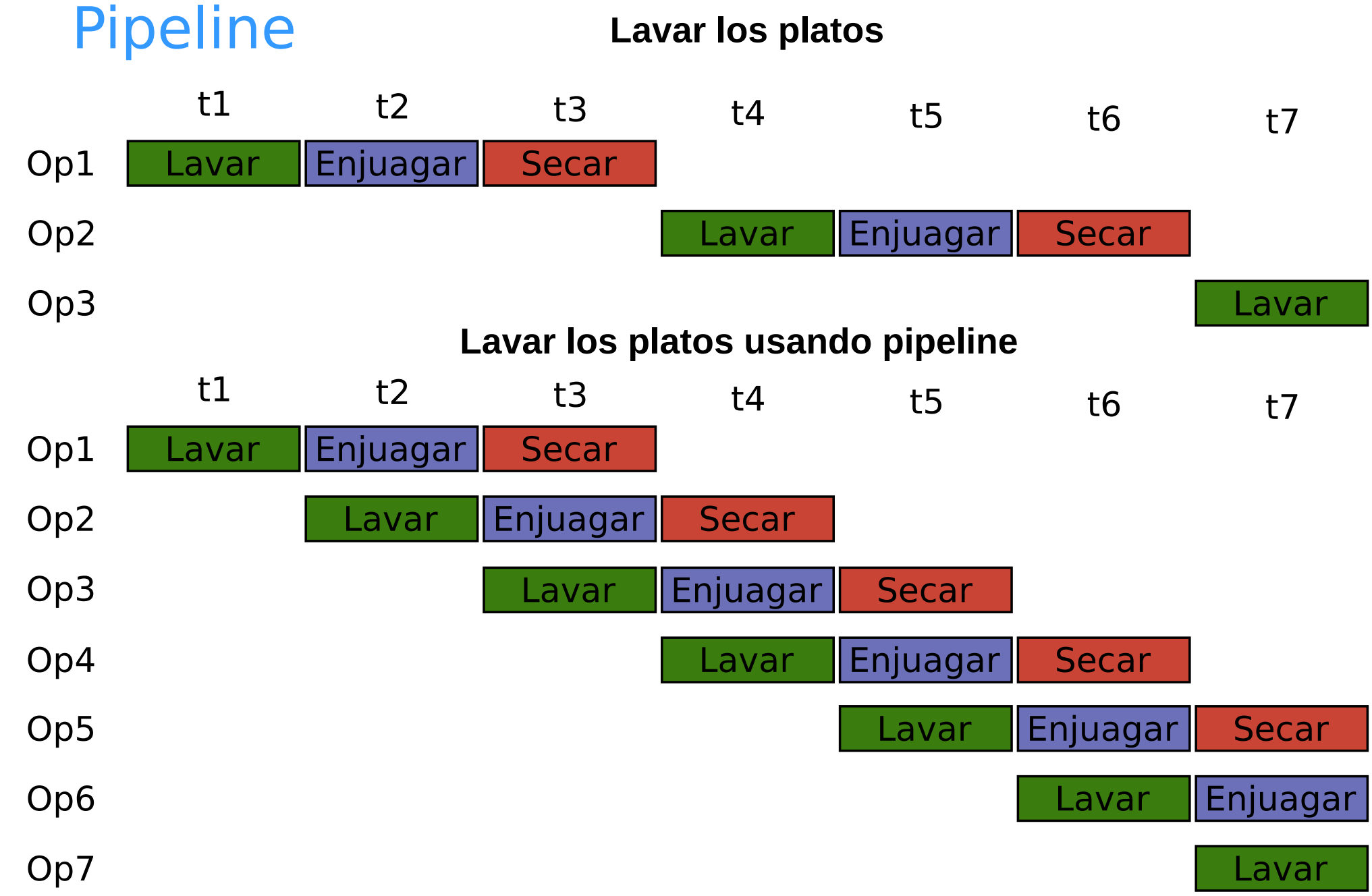
Bifurcación

- La bifurcación genera retraso en la ejecución del programa debido al corte en la ejecución secuencial de las instrucciones.
- En lo posible se tratan de evitar pero en algún punto del programa se vuelven imprescindibles.
- Mucho se ha escrito e investigado para predecir estos saltos, o evitarlos.
- Esto último principalmente con el uso de las ya vistas instrucciones condicionadas.

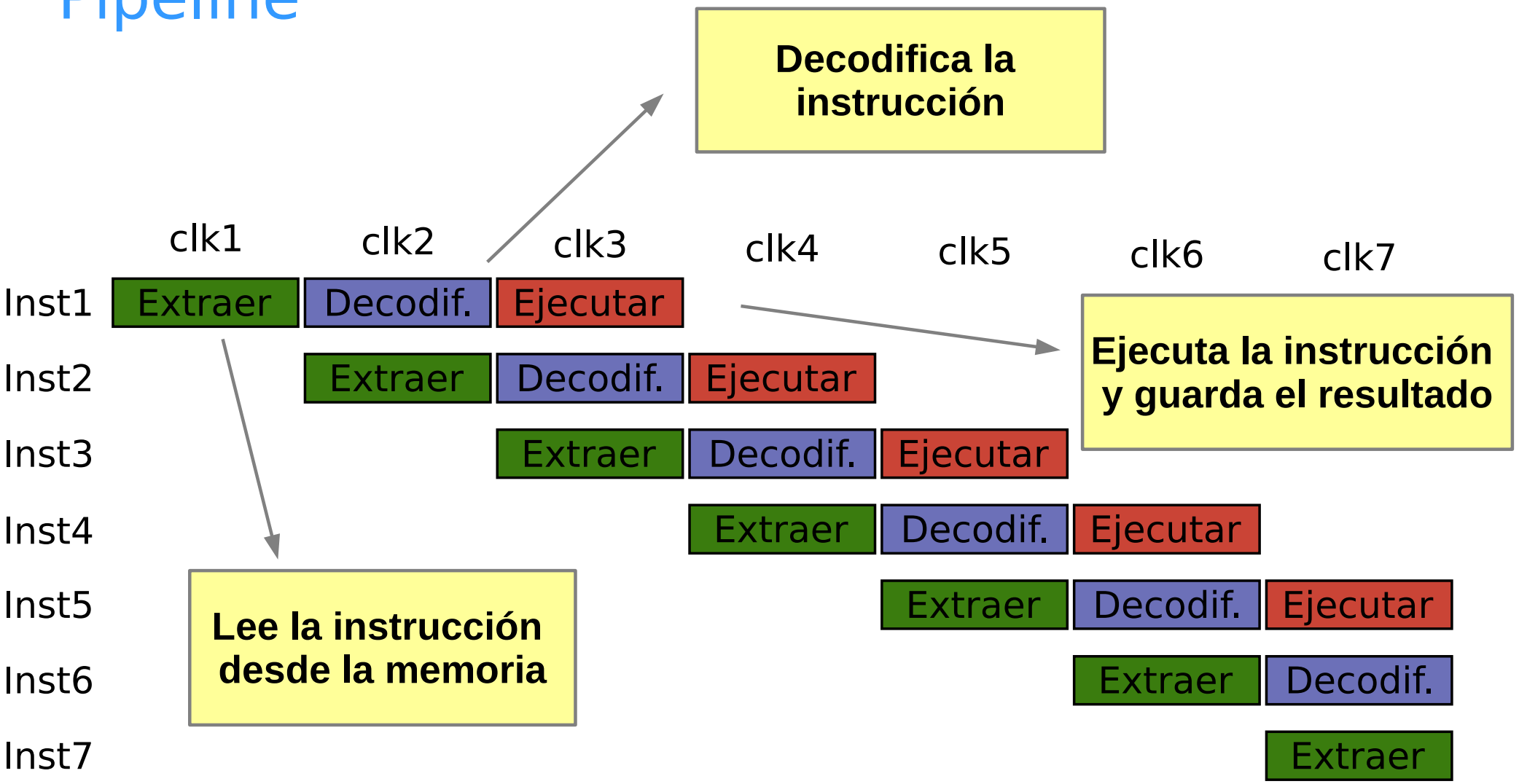
Pipeline

- Es una técnica de implementación mediante la cual varias instrucciones son superpuestas en la ejecución.
- Se fundamenta en paralelizar las acciones necesarias en la ejecución de una instrucción.
- Permite acelerar una CPU.
- Se puede entender como una linea de montaje, donde un producto requiere varios estaciones para ser montado, pero cada una de ellas opera en paralelo con las otras

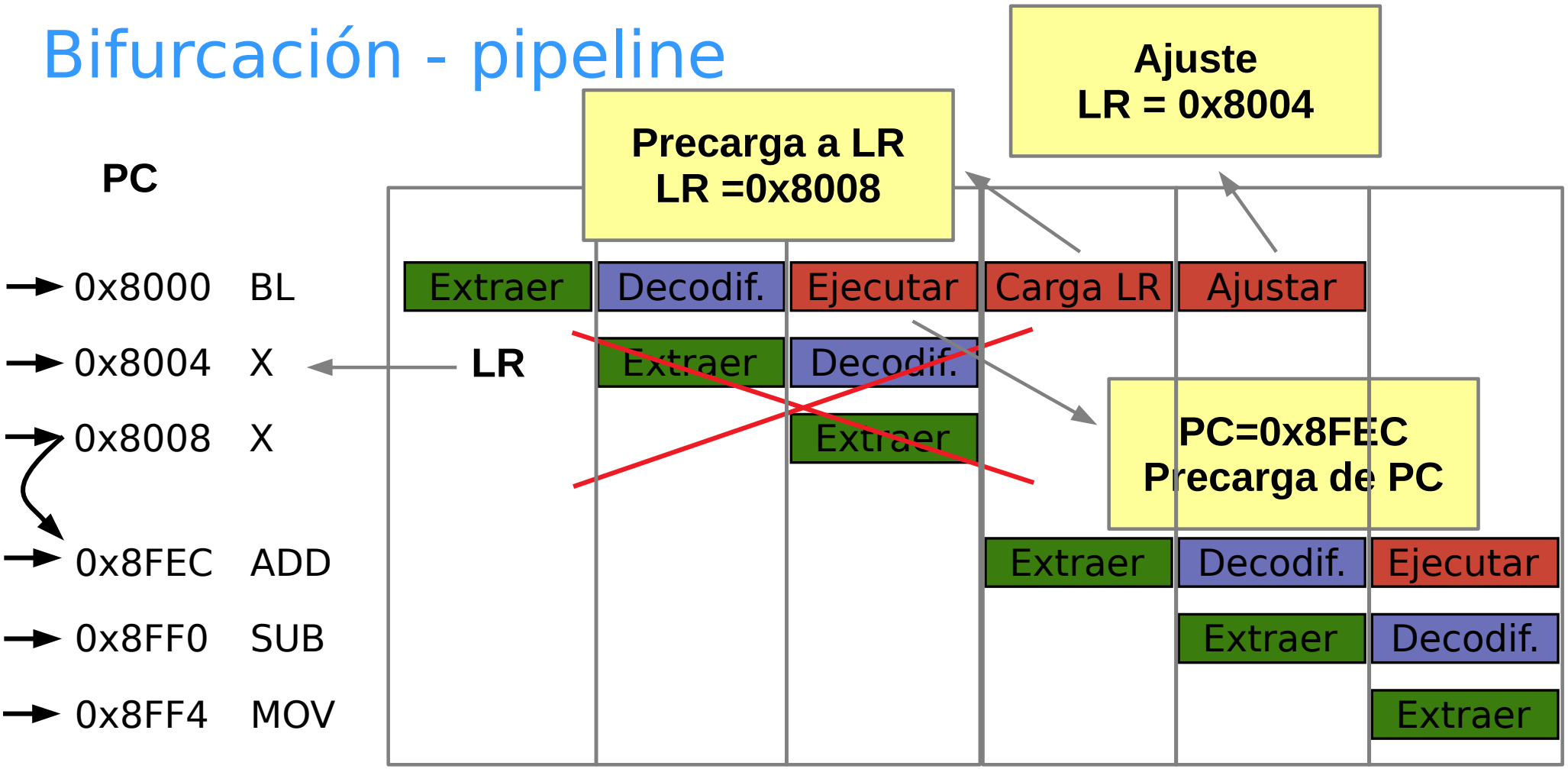
Pipeline


Lavar los platos
t1
t2
t3
t4
t5
t6
t7
Op1
Lavar
Enjuagar
Secar
Op2
Lavar
Enjuagar
Secar
Op3
Lavar
Lavar los platos usando pipeline
t1
t2
t3
t4
t5
t6
t7
Op1
Lavar
Enjuagar
Secar
Op2
Lavar
Enjuagar
Secar
Op3
Lavar
Enjuagar
Secar
Op4
Lavar
Enjuagar
Secar
Op5
Lavar
Enjuagar
Secar
Op6
Lavar
Enjuagar
Op7
Lavar

Pipeline


Decodifica la instrucción
clk1
clk2
clk3
clk4
clk5
clk6
clk7
Inst1
Inst2
Inst3
Inst4
Inst5
Inst6
Inst7
Extraer
Decodif.
Ejecutar
Ejecuta la instrucción y guarda el resultado
Lee la instrucción desde la memoria

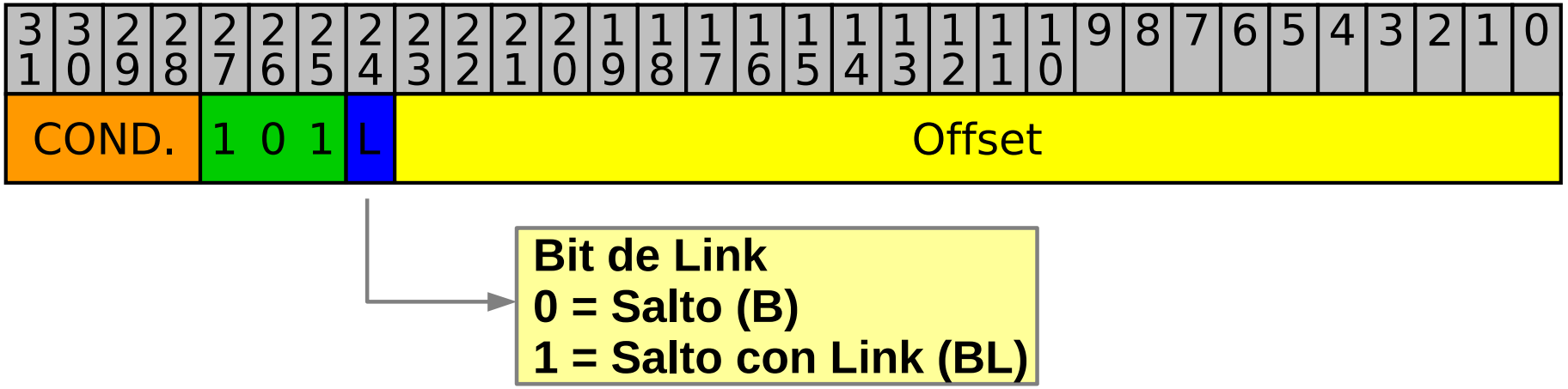
Bifurcación - pipeline



Bifurcación

- Un programa es usualmente ejecutado en secuencia.
- En arquitectura ARM, el contador de programa (PC) es automáticamente incrementado en 4 luego de cada instrucción para apuntar así a la próxima.
- Una instrucción de bifurcación o branch cambia el (PC) con la dirección donde se ubica la nueva instrucción a ejecutar.
- Tenemos dos tipos de salto **B** o salto simple y branch con link **BL**
- Además y como cualquier instrucción en ARM, el salto puede ser condicional o incondicional.

Salto



- La instrucción de Salto tanto **B** como **BL**, poseen un campo OFFSET de 24 bits
- El nro guardado en ese campo es sumado al PC cuando la instrucción es ejecutada.
- Para aumentar el rango se realiza una shift de 2 bit (todas las instrucciones ocupan 32 bits)
- Finalmente la instrucción **B** o **BL** posee un rango de **±32MB**

Salto

- Para superar la barrera de **±32MB** puedo operar directamente con el **PC.**

MOV PC,#0x0400000

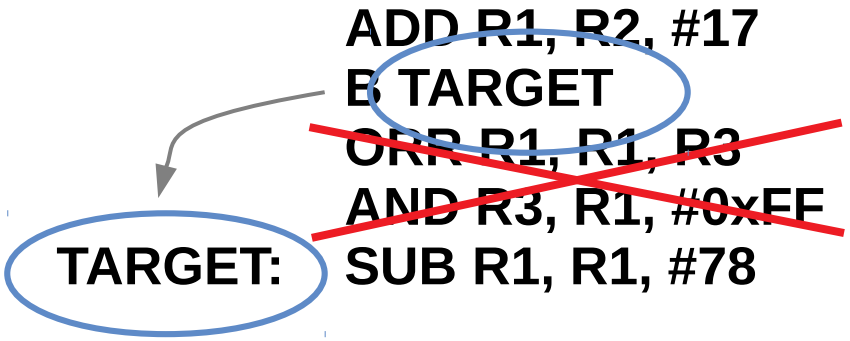
LDR PC,=0xBE00000

- El registro PC o R15 puede ser modificado como cualquier otro registro, esta permitido la asignación de una constante o incluso una operación aritmética.
- Otra opción es utilizar el valor de un registro como salto

BX R5

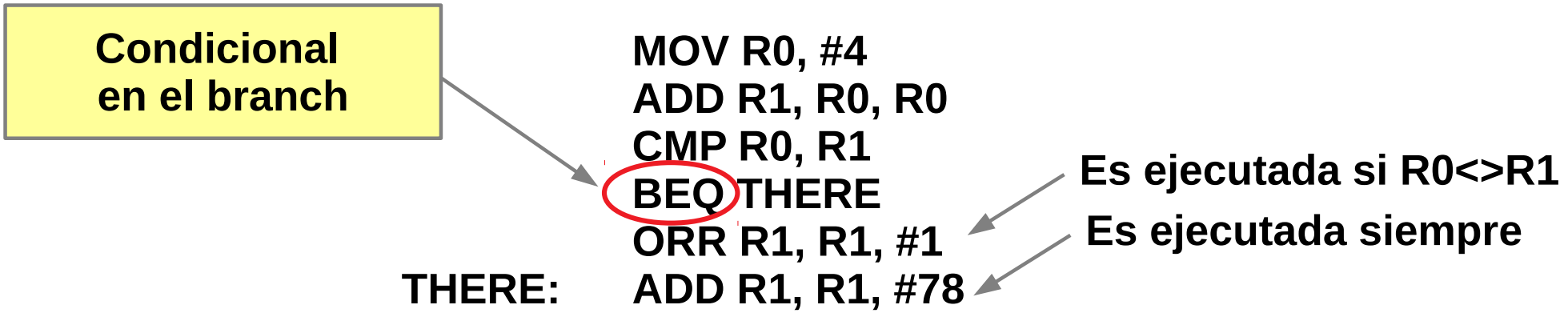
Salto Incondicional

- El siguiente código, muestra un salto incondicional utilizando la instrucción **B.**
- La instrucción siguiente a la ejecución de **B TARGET**, es la **SUB R1,R1,#78**, es decir aquella etiquetada **TARGET**.
- La etiqueta le da un nombre a la dirección de una instrucción, de esa manera puede ser indicada por una instrucción como **B**



Salto Condicional

- La instrucción **B** ahora es condicionada de acuerdo a si R0 es igual a R1.



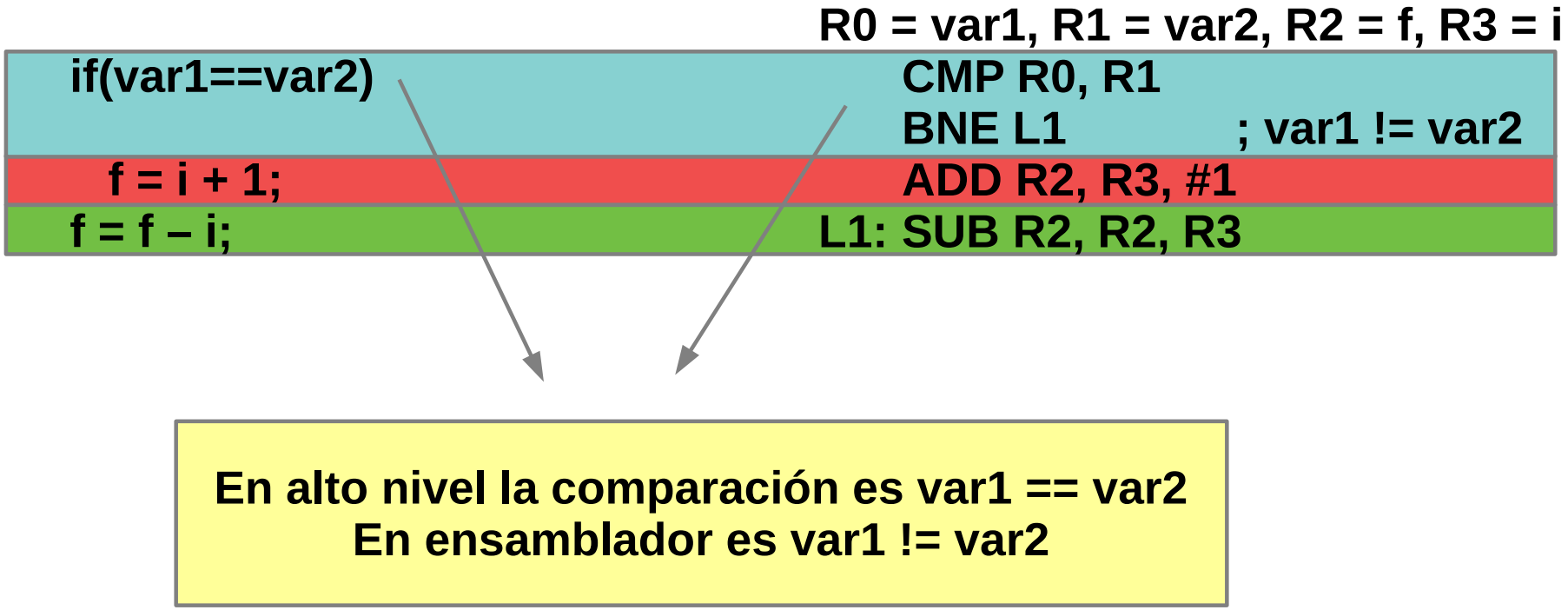
- Esto significa que si ambos registros son iguales Z=1 entonces la condicion EQ verifica el estado de esa bandera.
  - Si Z=1, el Salto es ejecutado y el programa salta a THERE.
  - Si Z=0, el Salto no es ejecutado y se procede a correr la instrucción siguiente.

Sentencia Condicional

- If, if/else, switch/case con sentencias condicionales que podemos encontrar en los lenguajes de alto nivel.
- Cada una de ellas permiten ejecutar una sentencia o grupo de ellas de acuerdo a una condición dada.
- A continuación se mostrará como trasladar estas sentencias de alto nivel a código ensamblador de ARM.

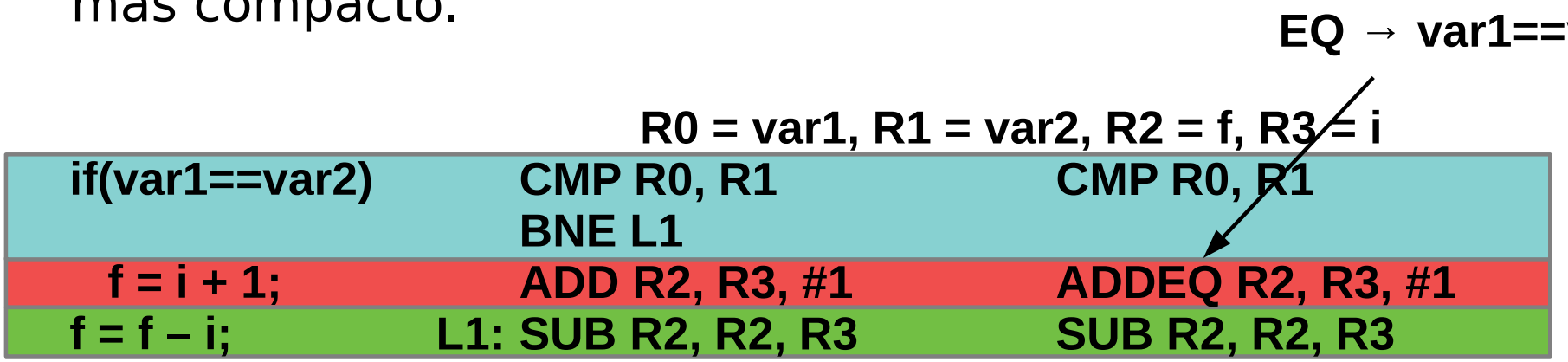
Sentencia **If**

- Una sentencia condicional if, ejecuta un bloque de instrucciones solo si la condición en if es cierta.



Sentencia **If**

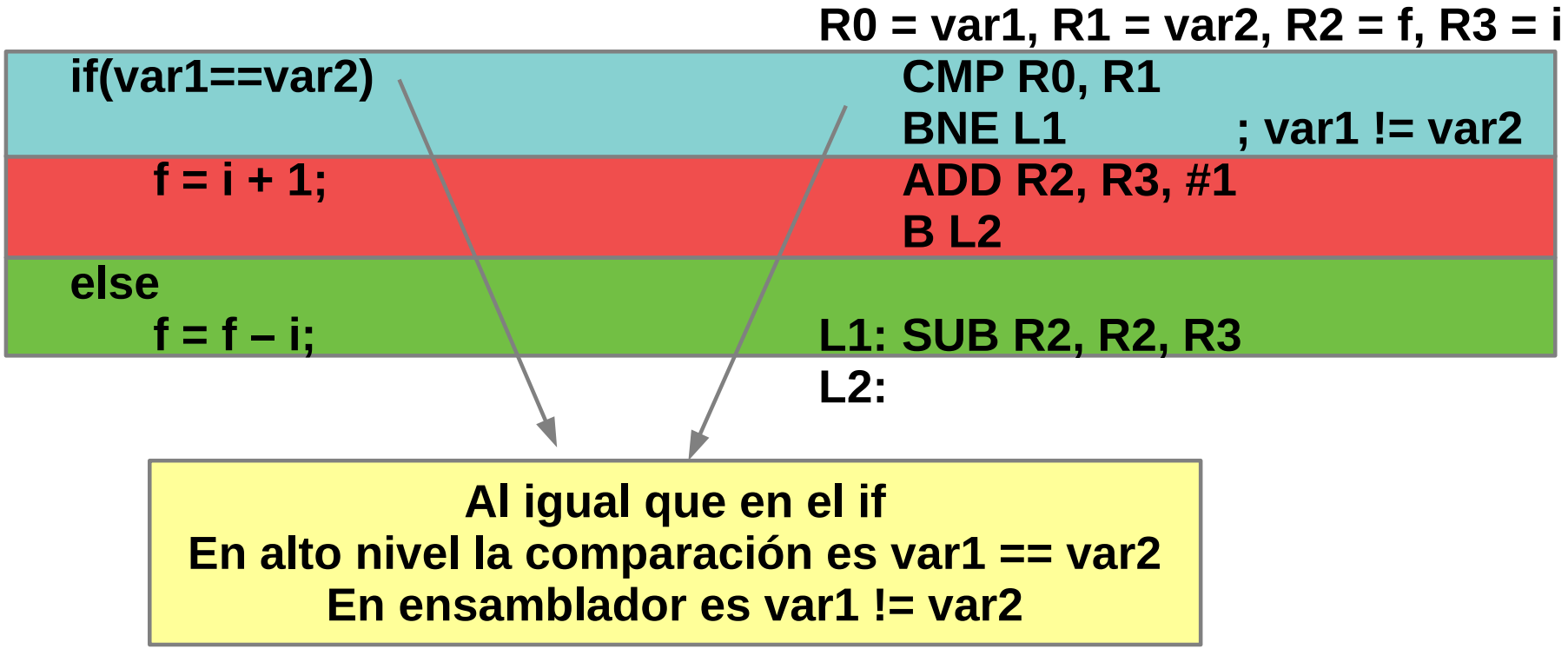
- Las instrucciones condicionadas, permite crear un código mas compacto.



- Ahora la operación condicionada es similar al ejemplo en alto nivel.
- Las instrucciones condicionadas, generan código mas compacto, mas rápido.
- Cuando el bloque de la condición es mayor a una o dos instrucciónes, es preferible la opción de salto condicionado.

Sentencia **If/else**

- La sentencia if/else, ejecuta uno de dos bloques de código, dependiendo de la condición.



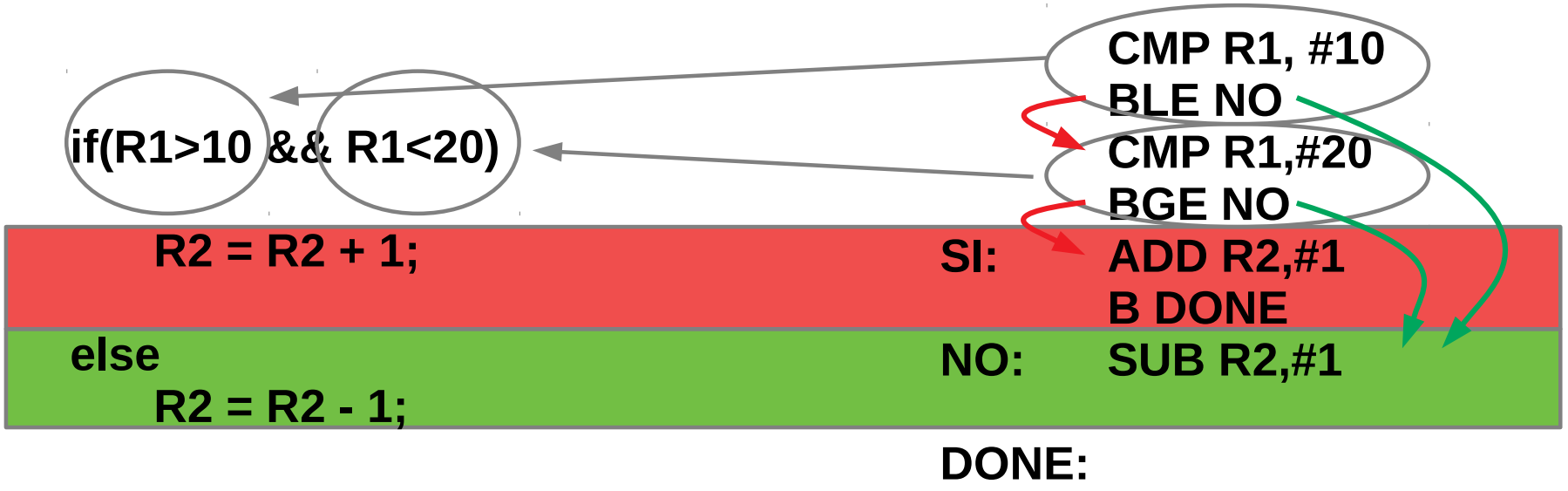
Sentencia **If/else**

- Igual que la sentencia if, el uso de instrucciones condicionadas, compacta mucho el código.

	R0 = var1, R1 = var2, R2 = f, R3 = i	
if(var1==var2)	**CMP R0, R1** **BNE L1**	**CMP R0, R1**
f = i + 1;	**ADD R2, R3, #1** **B L2**	**ADDEQ R2, R3, #1**
else		
f = f – i;	**L1: SUB R2, R2, R3**	**SUBNE R2, R2, R3**
	L2:	

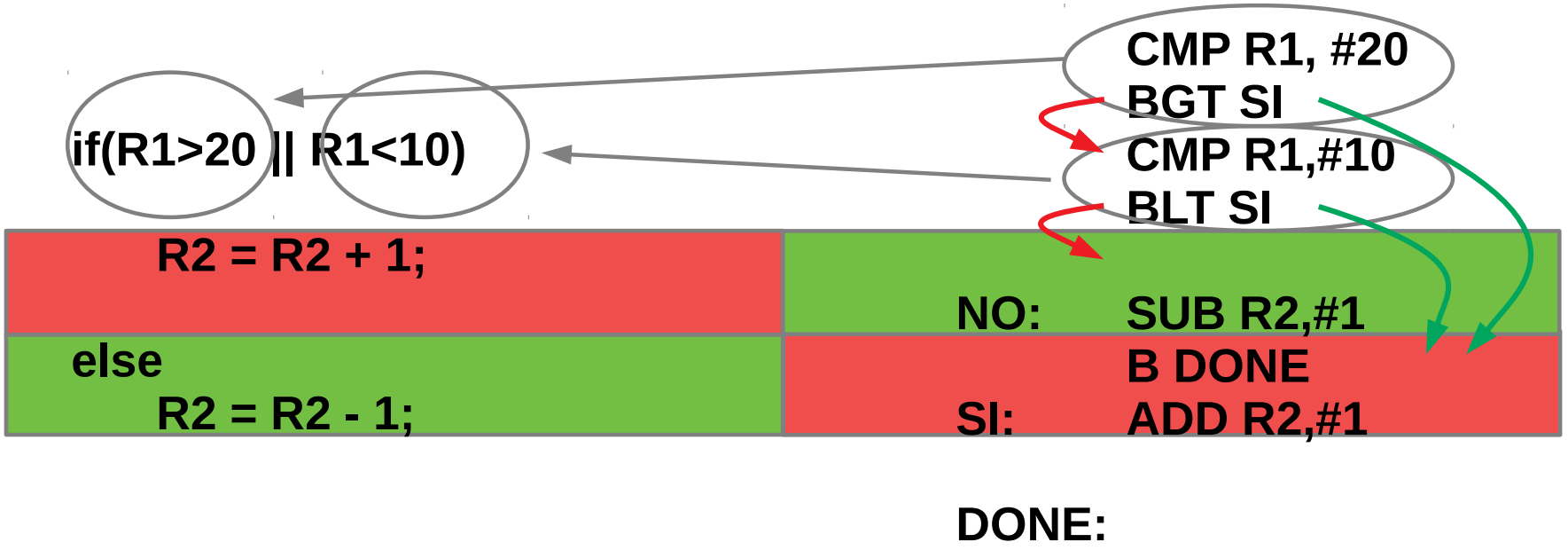
Sentencia **If/else** de múltiple condición.

- Una situación habitual es encontrarse con multiples condiciones.
- En el ejemplo dos condiciones unidas por una **AND**



Sentencia **If/else** de múltiple condición.

- Un ejemplo dos condiciones unidas por una **OR**



Sentencia **switch/case**

- Esta sentencia, permite ejecutar un bloque de varios, dependiendo de un condicional, si la condicional no se cumple para ningún bloque, se selecciona default.

switch(button) {	R0 = button, R1 = amt
case 1:	CMP R0, #1
amt=20;	MOVEQ R1, #20
break;	BEQ DONE
case 2:	CMP R0, #2
amt=50;	MOVEQ R1, #50
break;	BEQ DONE
case 3:	CMP R0, #3
amt=100;	MOVEQ R1, #100
break;	BEQ DONE
default: amt =0;	MOV R1, #0
}	DONE:

Loops o Bucles

- Permiten ejecutar un bloque repetidamente dependiendo de una condición.
- En lenguajes de alto nivel podemos ver dos tipos de bucles
  - Bucles While
  - Bucles For

Bucle While

- El bucle while, repite un código hasta que una condición no es cumplida.

Calcular x tal que $2^x=128$

```
int pow=1;
Int x=0;
while(pow!=128){

    pow=pow*2;
    x=x+1;
}
```

```
R0 = pow, R1 = x
    MOV R0, #1
    MOV R1, #0
WHILE:CMP R0, #128
    BEQ DONE
    LSL R0, R0, #1
    ADD R1, R1, #1
    B WHILE
DONE:
```

Igual que con if, la comparación en ensamblador es la inversa que el ejemplo en alto nivel

Bucle While

- El mismo bucle while, en ensamblador puede ser modificado levemente para igualar la condición con alto nivel.

Calcular x tal que $2^x=128$

```
int pow=1;
Int x=0;
while(pow!=128){
      pow=pow*2;
      x=x+1;
}
```

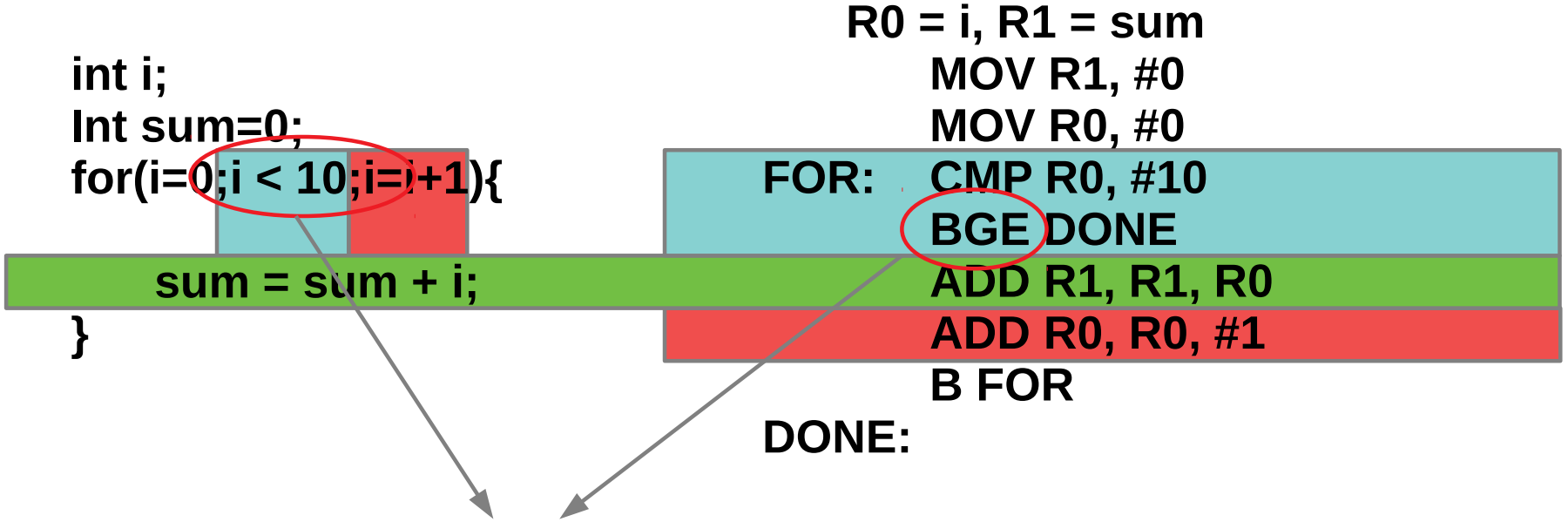
```
R0 = pow, R1 = x
      MOV R0, #1
      MOV R1, #0
      B TEST
WHILE:LSL R0, R0, #1
      ADD R1, R1, #1
TEST: CMP R0, #128
      BNE WHILE
```

Este modo, permite que la comparación en ensamblador sea igual que el ejemplo en alto nivel

Bucle for

- Este bucle clasico en lenguaje de alto nivel, combina:
  - Inicialización de variable.
  - Verificación de la condición.
  - El cambio de variable.

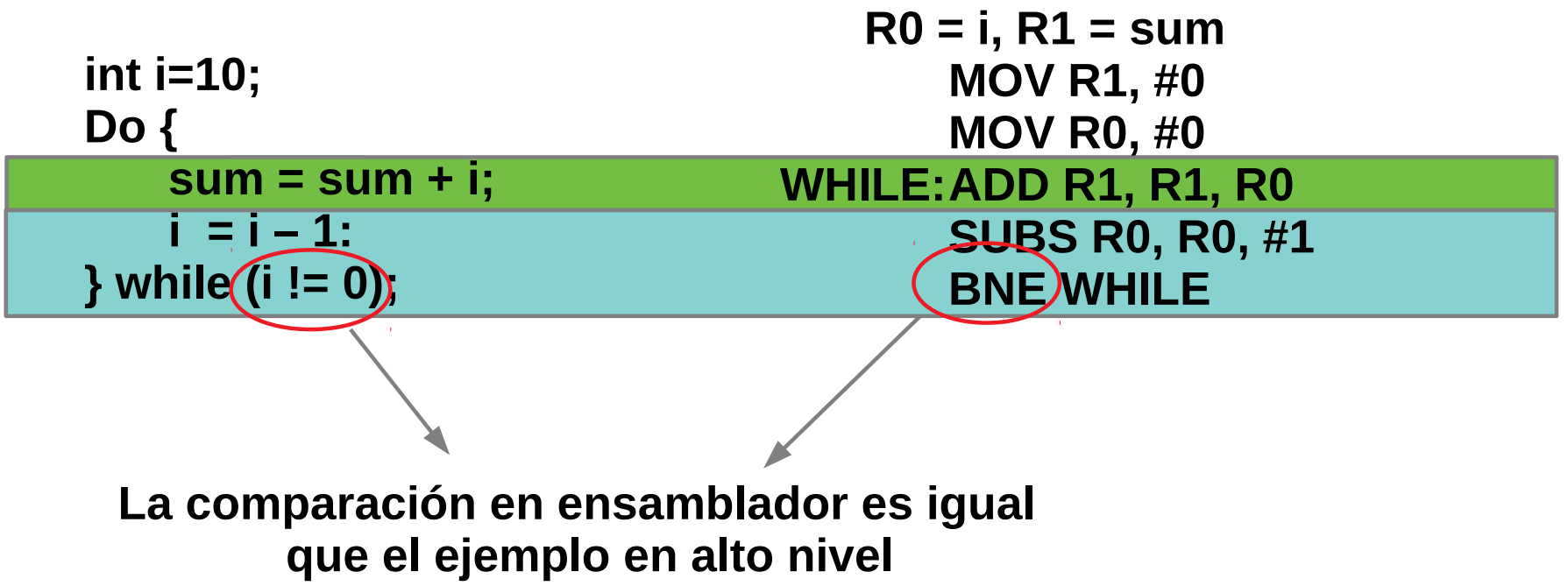
for(inicialización;condición;cambio)



La comparación en ensamblador (>=)
es la inversa que el ejemplo en alto nivel (<)

Bucle

- Se puede construir un bucle mas sencillo utilizando un solo branch.
- Este formato es similar al do while en lenguaje de alto nivel.
- Requiere que al menos una vez pase por el código



- Bibliografía

Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

William Hohl & Christopher Hinds. ARM assembly language. Fundamentals and techniques. 2nd edition. CRC press, 2015. Capítulo 8.

¿ Preguntas ?