IC-5701
Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

Agenda

TEC

- Repartición de registros
- Generación para Expresiones
- Generación para estructuras de control
- Generación para procedimientos

Generación de Código

Generación Código

- La fase final del compilador es la generación de código.
  - Programa destino equivalente en forma semántica
- Compiladores eficientes incluyen una fase de optimización antes y después de la generación de código.

Generación Código

- Generador de código tiene cuatro tareas principales:
  - Selección de instrucciones.
  - Repartición y asignación de registros
  - Manejo de la pila
  - Ordenamiento de instrucciones (saltos)
- Se particionan las instrucciones de representación intermedia en bloques básicos

Generación Código

```
[  op  |          target address          ]
    2                   1024                  decimal
 000010 00000 00000 00000 10000 000000   binary

[  op  |  rs |  rt | address/immediate]
   35     3     8           68           decimal
 100011 00011 01000 00000 00001 000100   binary
```

```
LD R0, a // R0 = a
ADD R0, R0, #1 // R0 = R0 + 1
ST a, R0 // a = R0
```

Entrada

- La entrada del generador de código es la representación intermedia producida por el front-end
  - Código tres direcciones
  - Cuádruplos
  - Tripletas
  - Representación de máquinas virtuales (bytecodes)

Programa Destino

- Influenciado por el conjunto de instrucciones de la máquina destino
- RISC (reduced instruction set computer)
  - Muchos registros, direccionamiento simple, conjunto de direcciones sencillo
- CISC (complex instruction set computer)
  - Pocos registros, instrucciones dos direcciones, diferentes modos de direccionamiento

Programa Destino

- En las máquinas basadas en pila, las operaciones se realizan metiendo operadores y operandos en la pila, y luego realizan las operaciones.
  - JVM
- Se puede producir programa objeto
  - Subprogramas se compilan por separado.
  - Enlace y carga

Programa Destino

- Producir un programa en lenguaje ensamblador facilita la generación de código.
- Se generan instrucciones simbólicas y se utilizan las macros del ensamblador.
  - Desventaja de paso de ensamblador a máquina.

Selección de instrucciones

- Complejidad:
  - Nivel de representación
  - Arquitectura de instrucciones
  - Calidad
- Uniformidad y precisión
- Velocidades de las instrucciones y las características específicas

Asignación de registros

- Repartición de registros
  - Seleccionar que variables residen en registros en cada bloque.
- Asignación de registros
  - Asignar registros a cada variable en cada bloque.

Lenguaje Destino

Instrucciones

- Operaciones de carga:
  - load, LD, carga un valor de una ubicación en un registro
  - `lw, lb, li, l.s, li.s`
- Operaciones de almacenamiento
  - store, ST, almacena un valor de un registro en una ubicación
  - `sw, sb, s.s`

Instrucciones

- Operaciones de cálculo:
  - Realizar el cálculo de aplicar un operador (ADD, SUB, MULT) a dos valores y colocar el resultado en un registro
  - `add, subu, add.s`
- Saltos incondicionales
  - Se traslada el control hacia una etiqueta del programa destino
  - `j, jr`

Instrucciones

- Saltos condicionales
  - Se traslada el control hacia una etiqueta del programa destino si se cumple una condición (expresión condicional).
  - `slt $t0, $t0, $t1`
  - `beq $t0, 0, _Label10`
  - `c.lt.s`

Modos de direccionamiento

- Espacio de memoria reservada para una variable.
  - `res:    .asciiz "La suma de los numeros es: ”`
  - `array1: .byte   'a','b'`
- Una ubicación también puede ser una dirección indexada de la forma a(r)
  - `lw    $t2, 4($sp)`
  - `sw    $t2, -12($sp)`

Modos de direccionamiento

- Una ubicación de memoria puede ser un entero indexado por un registro
  - `lw $t1, ($t0)`
  - `sw $t1, 4($t0)`
- Direccionamiento constante inmediato
  - `li $t1, #5`
  - `li $t1, #100`

Modos de direccionamiento

- En la generación de código se utilizan las técnicas de heurística.
- El costo de una instrucción será de 1 + más los costos del direccionamiento de los operandos.
- Costo direccionamiento registro 0
- Costo direccionamiento memoria 1

Bloques básicos

Bloques básicos

- El flujo de control sólo puede entrar en el bloque básico a través de la primera instrucción.
  - No hay saltos
- El control saldrá del bloque sin detenerse o bifurcarse.
  - Excepto tal vez en la última instrucción.

TEC

Bloques básicos

- Se debe separar el código de tres direcciones en bloques básicos para facilitar la generación de código.
- Se inicia con la primera línea y se agrega hasta encontrar un salto o etiqueta.

Instrucciones Líderes

- La primera instrucción de tres direcciones en el código intermedio es líder.
- Cualquier instrucción que sea el destino de un salto condicional o incondicional es líder.
- Cualquier instrucción que siga justo después de un salto condicional o incondicional es líder

Bloques básicos

- Para cada instrucción líder, su bloque básico consiste en sí misma y en todas las instrucciones hasta, pero sin incluir, la siguiente instrucción líder o el final del programa intermedio.

Bloques básicos

```
1)  i = 1
2)  j = 1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j+1
9)  if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
```

```
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)
```

Bloques básicos

```
1)  i = 1
2)  j = 1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j+1
9)  if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
```

```
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)
```

Actividad #4

```
1)  t1 = j
2)  t2 = n
3)  t3 = t1 < t2
4)  if(t3) goto (6)
5)  goto (14)
6)  t4 = k
7)  t5 = t1 * 2
8)  t6 = t4 + t5
9)  k = t6
10) m = t5
11) t7 = t1 + 1
12) j = t7
13) goto (1)
14)
```

Actividad #4

TEC

```
t1 = b
t2 = t1 >= 10
if t2 goto _main_if_1_if
goto _main_if_1_if_end
_main_if_1_if:
_main_for_1_begin:
dataInt i
i = 0
_main_for_1_condicion:
t3 = i
t4 = b
t5 = t3 <= t4
if t5 goto _main_for_1_bloque
goto _main_for_1_end
_main_for_1_alter:
t9 = i
t10 = t9 + 1
i = t10
goto _main_for_1_condicion
_main_for_1_bloque:
t6 = a
param t6
t7 = b
param t7
t8 = call potencia, 2
p = t8
goto _main_for_1_alter
_main_for_1_end:
goto _main_if_1_end
_main_if_1_if_end:
_main_if_1_end:
```

Repartición y asignación de registros

Generación Código

- La fase final del compilador es la generación de código.
  - Programa destino equivalente en forma semántica
- Compiladores eficientes incluyen una fase de optimización antes y después de la generación de código.

Generación Código

- Generador de código tiene tres tareas principales:
  - Selección de instrucciones.
  - Repartición y asignación de registros
  - Ordenamiento de instrucciones
- En la generación de código se utilizan las técnicas de heurística.

Programa Destino

- Producir un programa en lenguaje ensamblador facilita la generación de código.
- Se generan instrucciones simbólicas y se utilizan las macros del ensamblador.

Manejo de registros

- Las operaciones que sólo usan registros son más rápidas.
  - La velocidad del procesador es más rápida que la velocidad de memoria
- Qué debe residir en registros y en qué registro.

Repartición de registros

- Se debe reservar algunos para guardar los valores más activos.
  - Variables globales
  - Los valores pueden ser distintos en varios ciclos (procedimientos)
- Los registros restantes se utilizan para cálculos temporales y variables locales.

Repartición de registros

Número de registro	Nombre Alternativo	Descripción
0	0	Valor 0
1	$at	Reservado por ensamblador
2-3	$v0-$v1	Evaluación de expresiones y resultado de funciones (valores)
4-7	$a0-$a3	Primeros cuatro parámetros de subrutinas (argumentos)
8-15	$t0-$t7	Manejo de valores temporales (temporales)

Generación para estructuras de control

Estructuras de Control

- La generación de código para estructuras de control involucra la generación de etiquetas.
  - Similar a variables temporales en 3Direcciones
  - Se utilizan para saltos
- Incluyen if, while, for, do-while, break, switch

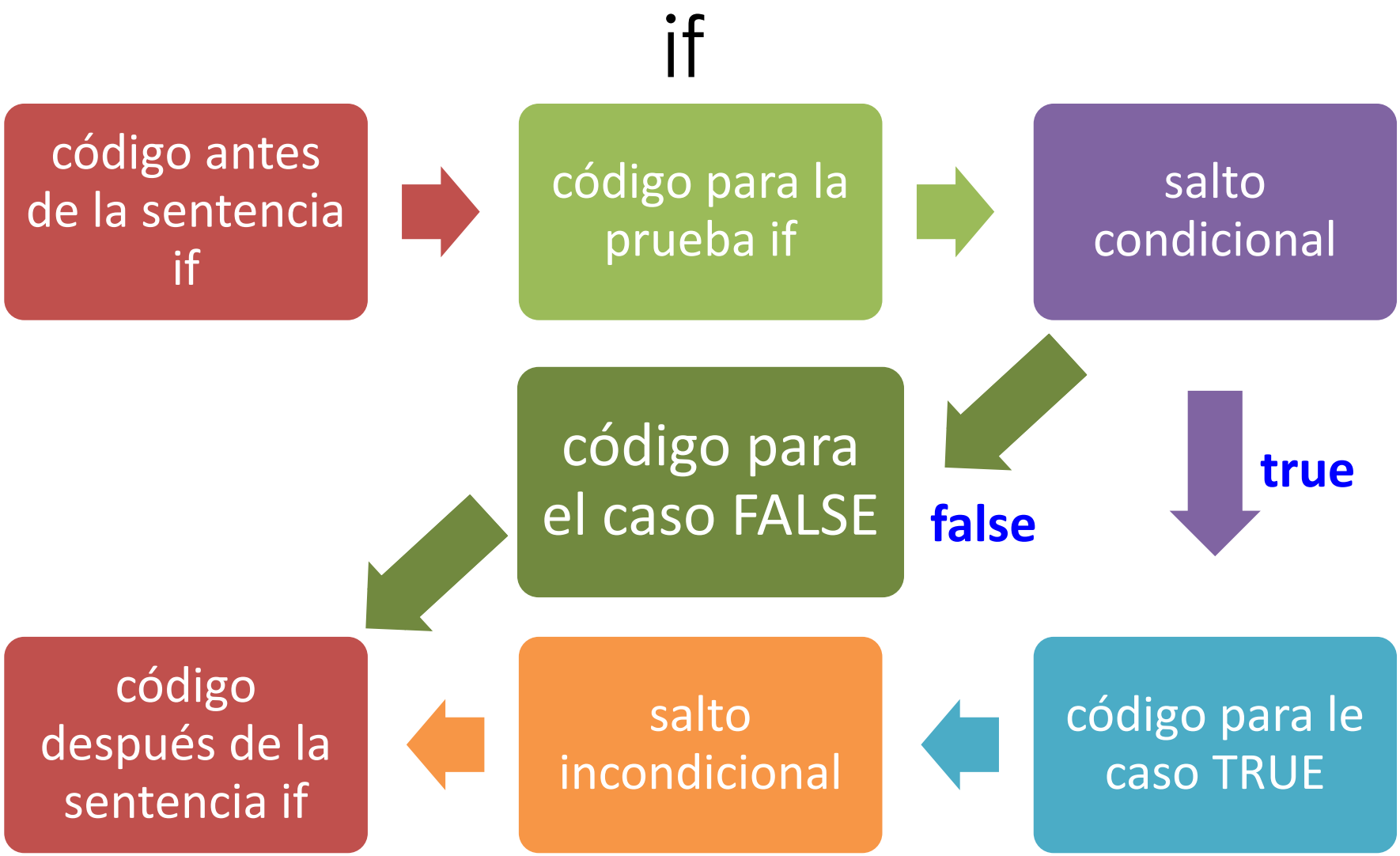
if, while & for

- Se debe traducir de control estructurado a uno no estructurado (saltos).
- Los compiladores tratan de hacer uso eficiente de los saltos.

if, while & for

- Las condiciones verdaderas no necesitan saltos.
  - Reduce el número de saltos
- Saltos condicionales e incondicionales

TEC


if
código antes de la sentencia if
código para la prueba if
salto condicional
true
false
código para el caso FALSE
código para le caso TRUE
salto incondicional
código después de la sentencia if

TEC

if

```
if ( E ) S1 else S2
```

```
<código para evaluar E a t1>
if_false t1 goto L1
<código para S1>
goto L2
label L1
<código para S2>
label L2
```

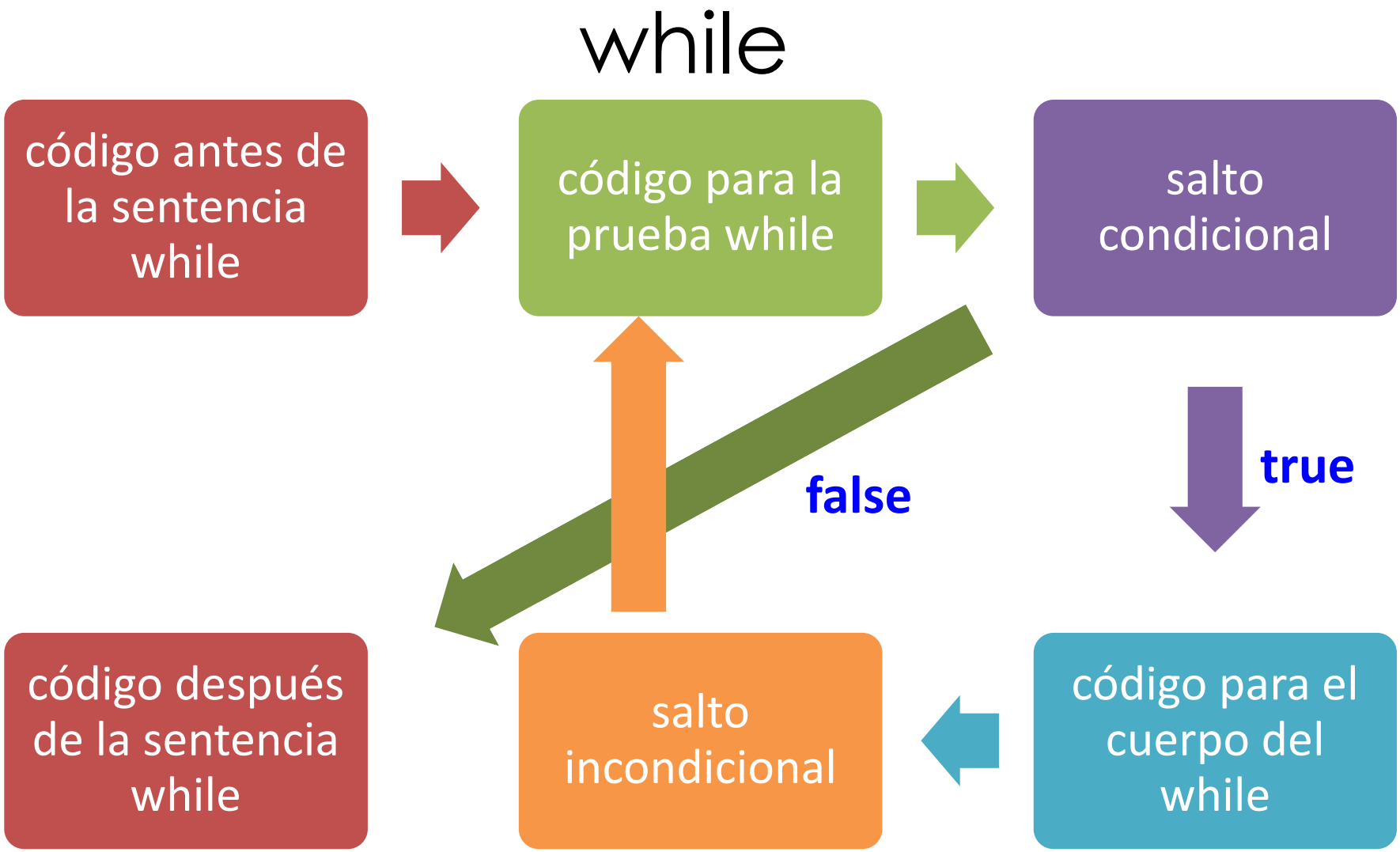
TEC

if

```
if ( E ) S1 else S2
```

```
<código para evaluar E a $t7>
beq  $t7,0,_L1
<código para S1>
j _L2
_L1:
<código para S2>
_L2:
```

TEC


while
código antes de la sentencia while
código para la prueba while
salto condicional
true
false
código después de la sentencia while
salto incondicional
código para el cuerpo del while

TEC

while

```
while ( E ) S
```

```
label L1
<código para evaluar E a t1>
if_false t1 goto L2
<código para S>
goto L1
label L2
```

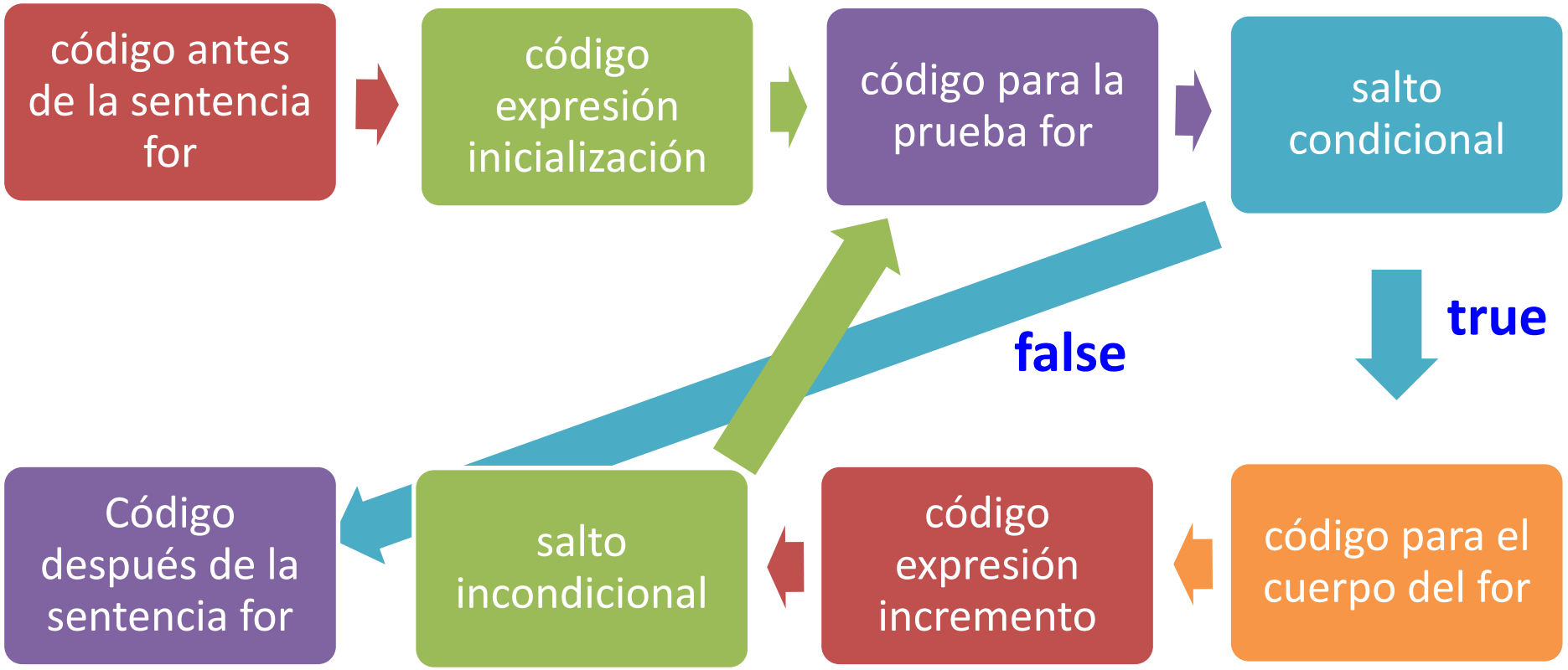
TEC

while

```
while ( E ) S
```

```
_L1:
<código para evaluar E a $t7>
beq  $t7,0,_L2
<código para S>
j _L1
_L2:
```

for


código antes de la sentencia for
código expresión inicialización
código para la prueba for
salto condicional
true
false
código para el cuerpo del for
código expresión incremento
salto incondicional
Código después de la sentencia for

for

```
for( E1; E2; E3 ) S
```

```
<código para evaluar E1>
label L1
<código para evaluar E2 a t1>
if_false t1 goto L2
<código para S>
<código para evaluar E3>
goto L1
label L2
```

TEC

for

```
for( E1; E2; E3 ) S
```

```
<código para evaluar E1>
_L1:
<código para evaluar E2 a $t7>
beq  $t7,0,_L2
<código para S>
<código para E3>
j _L1
_L2:
```

Generación para procedimientos

TEC

Opcional

- Leer y resumen:
  - Resumen Lectura Generación de código

Actividad #4

```
#int main()
#{
#    int int1#
#    float float1#
#    print("\nIndique un numero: ")#
#    read(int1)#
#    print("\nIndique un flotante: ")#
#    read(float1)#
#    print("\n")#
#    print(int1)#
#    print("\n")#
#    print(float1)#
#    print("\nAdiós!")#
#}
```

Ejercicio

- Genere el código mips para el siguiente código fuente

```
int main ()
{
        int num = -1;
        string str = “Hola”;
        print(num);
        print(str);
}
```

Ejercicio

```
# Ejercicio SPIM

        .data
hola:   .asciiz "\nHola\n"
        .globl  main

        .text
main:

    # inicializar
    li    $t0, -1

    # imprimir valor num
    li    $v0, 1
    move  $a0, $t0
```

Bibliografía

- Programming Language Processors in Java: compilers and interpreters. Watt, David, Brown, Deryck. Pearson Education. 2000

- Compilers: principles, techniques and tools (2da. ed.). Aho, Alfred. Pearson Education. 2007


TEC
Tecnológico
de Costa Rica