

Tema 6. Ejercicios.

6.1 Dados los siguientes programas:

programa (a)	programa (b)	programa (c)
(1) prod := 0	(1) j:=n	(1) x := y;
(2) i := 1	(2) t1:= 4*n	(2) x := x + 1;
(3) t1 := 4 * i	(3) v:=a[t1]	(3) z := x + 5;
(4) t2 := a[t1]	(4) j:=j+1	(4) if x < 20 goto (2);
(5) t3 := 4 * i	(5) t2:= 4*j	(5) v := z + 1;
(6) t4 := b[t3]	(6) t3:=a[t2]	(6) goto (9);
(7) t5 := t2 * t4	(7) if j>v goto (9)	(7) v := v + 1;
(8) t6 := prod + t5	(8) goto (11)	(8) if v < 10 goto (10);
(9) prod := t6	(9) x:=t3	(9) goto (7);
(10) t7 := i + 1	(10) goto (4)	(10) x := 0;
(11) i := t7	(11) x:=t3	
(12) if i <= 20 goto (3)		

Especifica cuáles son los líderes y construye el grafo de flujo.

6.2 El núcleo del método para asociar la información sobre *próximo uso* a una secuencia de instrucciones está descrito a continuación. Aplicar el método a las secuencias de instrucciones descritas abajo, reflejando, paso a paso, la evolución del contenido de la tabla de símbolos. Todas las variables están vivas al finalizar el bloque. Visto el resultado, ¿se podría decir que hay código muerto?

Si la instrucción *i* tiene la forma ***x := y op z***, entonces

1. Asociar a la instrucción *i* la información en curso en la tabla de símbolos sobre ***y***, ***x*** y ***z***.
2. Marcar ***x*** en la tabla de símbolos como "muerto".
3. Marcar ***y*** y ***z*** como "vivos" e introducir *i* como próximo uso de ***y*** y ***z***.

programa (a)	programa (b)	programa (c)
(1) b:=b*a	(1) d := c	(1) d := a + b
(2) b:=a*a	(2) e := a - b	(2) c := d + c
(3) a:= - a	(3) f := a - c	(3) e := a + b
(4) b:=d	(4) d := e + f	(4) c := e + c
(5) a:=b+c	(5) d := d + f	(5) f := a + b
(6) c:= c * b		(6) c := f

Algoritmo de generación de código Para cada instrucción de la forma x := y op z (similar para x := op y) se realizan las siguientes acciones:			
1. Llamar a la función obtenreg , para determinar la localización L sobre la que se realizará el cálculo de y op z. Habitualmente, L será un registro, aunque puede ser una dirección de memoria.			
2. Consultar el Descriptor de direcciones para y con el objetivo de determinar y', una de las localizaciones en curso de y (*). Si el valor de y no se encuentra en L, generar la instrucción MOV y',L para obtener una copia de y' en L.			
3. Generar la instrucción op z' , L , donde z' es una de las localizaciones en curso de z (*). Modificar el valor del descriptor de direcciones de x, indicando que su valor actual se encuentra en L. Si L es un registro, modificar su descriptor para indicar que contiene el valor de x.			
4. Si el valor de y y/o z no tiene/n más usos, esto es, no están vivos a la salida del bloque, alterar la descripción de los registros que contengan los valores de y y/o z, para indicar que sus valores no volverán a ser utilizados.			
(*)Si el valor de y se encuentra en un registro y en memoria, se escogerá el registro.			
obtenreg Devuelve la localización L donde quedará el valor de x tras la ejecución de la instrucción x := y op z . Este algoritmo se basa en la información sobre próximo uso:			
1. Si el valor de y se encuentra en un registro que no contiene el valor de otras variables, e y no está viva después de la ejecución de x := y op z , entonces devolver y. Modificar el descriptor de y para indicar que y no se encuentra a partir de este momento en L.			
2. Si falla 1, devolver el primer registro vacío, caso de que exista alguno.			
3. Si falla 2, si x tiene algún uso más en el bloque, (o bien op exige un registro) buscar un registro ocupado R. Guardar el valor de R en memoria si no se encuentra ya en ella (MOV R, M). Modificar el descriptor de memoria para M, y devolver R. Si R contenía el valor de varias variables, será necesaria una instrucción MOV para cada una de ellas.			
4. Si x no se usa en el resto del bloque, devolver la localización de x.			

6.3 Se ha aplicado el algoritmo de generación de código para el código de tres direcciones que aparece más abajo. El código generado, sin embargo, no es correcto, es decir, no se ha aplicado correctamente el algoritmo de generación de código. Encuentra el error en la columna de nombre **código generado** y razona por qué se ha aplicado incorrectamente el algoritmo.

Instrucción	Código generado	Desc. regs.	Desc. direcciones
		R0:a R1:c	b, d en memoria a en R0, c en R1
1) b := c + a (b,v,2) (c,v,3) (a,v,2)	ADD R0, R1	R0:a R1:b	c, d en memoria a en R0, b en R1
2) d := b + a (d,v,3) (a,m) (b,m)	ADD R0, R1	R0:vacio R1:d	c en memoria d en R1
3) b := c + d (b,v,4) (c,m) (d,v,5)	MOV c, R0 ADD R1, R0	R0:b R1:d	b en R0 d en R1
4) a := b (a,v,5) (b,v,5)		R0:a,b R1:d	a,b en R0 d en R1

- 6.4** Se ha aplicado el algoritmo de generación de código para el código de tres direcciones que aparece más abajo. Al examinar la primera instrucción una alumna dice que el algoritmo genera la instrucción que aparece en la tabla. Responde, razonando la respuesta, si te parece que la alumna tiene razón o está equivocada.

Instrucción	Código generado	Desc. regs.	Desc. direcciones
		R0:a R1:c,d	b, c en memoria a en R0, c,d en R1
1) b := c + a (b,v,2) (c,m) (a,v,2)	ADD R0, R1		

- 6.5** Aplicar el algoritmo de generación de código partiendo de la situación inicial descrita.

Instrucción	Código generado	Desc. regs.	Desc. direcciones
		R0:a R1:c	b, c, d en memoria a en R0, c en R1
1) b := a + c (b,v,2) (c,v,3) (a,v,2)			
2) d := b + a (d,v,3) (a,m) (b,m)			
3) b := c + d (b,v,4) (c,v,5) (d,v,5)			
4) a := b (a,v,5) (b,v,5)			

Instrucción	Código generado	Desc. regs.	Desc. direcciones
		R0:a R1:c	b, d en memoria a en R0, c en R1
1) b := c + a (b,v,2) (c,v,3) (a,v,2)			
2) d := b + a (d,v,3) (a,m) (b,m)			
3) b := c + d (b,v,4) (c,m) (d,v,5)			
4) a := b (a,v,5) (b,v,5)			