Problema

Dado el siguiente fragmento de código:

```
for (i=0; i<100; i++) {
     if (A[i]>20) then {
          X[i]:=1;
     } else {
          if (A[i]>10) then {
               X[i]:=2;
          } else {
               X[i]:=3;
          }
     }
}
```

a) Genere el código intermedio. Considere que los vectores `A` y `X` se encuentran almacenados en las posiciones de memoria almacenadas en los registros R10 y R20, respectivamente, que todas las operaciones se realizan con registros enteros y que el contenido del registro R0 siempre es 0.

b) A partir del código que ha obtenido en el apartado (a), genere el código intermedio equivalente aplicando operaciones con predicado al mayor número de instrucciones que le sea posible.

c) Optimice el código del apartado (b) todo lo que le sea posible. ¿En qué porcentaje ha reducido el número de instrucciones ejecutadas en comparación con el código del apartado (b)?

d) Genere el código VLIW correspondiente a los apartados (a) y (c) para un procesador VLIW con instrucciones dotadas de 3 slots: uno para operaciones de carga/almacenamiento (2 ciclos de latencia), un segundo para las operaciones en coma flotante (3 ciclos de latencia) y un tercero para los operaciones enteras/saltos (1 ciclo de latencia).

Solución

a) Una posible solución se muestra continuación:

```
          ADDI R1,R0,#1          // R1:= 10
          ADDI R2,R0,#2          // R2:= 20
          ADDI R3,R0,#3          // R3:= 30
inicio:   LD   R5,0(R10)         // R5:= A[i]
          SUBI R6,R5,#20         // R6:= R5-20
          BLT  R6,else           // if (R6 <=0) go to else
          SD   0(R20),R1         // X[i]:= 10
          JMP  final             // go to final
else:     SUBI R6,R5,#10         // R6:= R5-10
          BLT  R6,else2          // if (R6 <=0) go to else2
          SD   0(R20),R2         // X[i]:= 20
          JMP  final             // go to final
else2:    SD   0(R20),R3         // X[i]:= 30
final:    SUBI R10,R10,#4        // R10:= R10-4
          SUBI R20,R20,#4        // R20:= R20-4
          BNEZ R10,inicio        // if (R10 > 0) go to inicio
```

b) A partir del código del apartado anterior, se puede obtener el siguiente código con predicados:

```
inicio:   LD        R5,0(R10)
          PRED_CL   p3,p4                    // p3:=0; p4:=0
          PRED_LT   p1,p2,R5,#20             // if (A>20) {p2:=1; p1:=0}
          PRED_LT   p3,p4,R5,#10   (p1)      // if (A>10) {p4:=1;p3:=0}
          ADDI      R1,R0,#1       (p2)
          SD        0(R20),R1      (p2)
          ADDI      R2,R0,#2       (p4)
          SD        0(R20),R2      (p4)
          ADDI      R3,R0,#3       (p3)
          SD        0(R20),R3      (p3)
          SUBI      R10,R10,#4
          SUBI      R20,R20,#4
          BNEZ      R10,inicio
```

Se han obtenido 13 instrucciones por lo que se ejecutarán 1300 instrucciones para procesar un vector de 100 elementos.

c) Una posible optimización del código con predicados sería el siguiente:

```
inicio:   LD        R5,0(R10)
          PRED_CL   p3,p4                // p3:=0; p4:=0
          PRED_LT   p1,p2,R5,#20         // if (A>20)  {p2:=1; p1:=0}
          PRED_LT   p3,p4,R5,#10   (p1)  // if (A>10)  {p4:=1; p3:=0}
          ADDI      R1,R0,#1       (p2)
          ADDI      R1,R0,#2       (p4)
          ADDI      R1,R0,#3       (p3)
          SD        0(R20),R1
          SUBI      R10,R10,#4
          SUBI      R20,R20,#4
          BNEZ      R10,inicio
```

Lo que se ha optimizado es el acceso a memoria ya que se han eliminado dos instrucciones de almacenamiento. Ahora el total de instrucciones que se tiene es de 11 lo que supone una ejecución de 1100 instrucciones para el procesamiento de un vector de 100 elementos.

d) Código VLIW para el apartado (a).

	Operaciones de carga/almacenamiento	Operaciones en coma flotante	Operaciones enteras y saltos	
	-------	-------	ADDI R1,R0,#1	1
		-------	ADDI R2,R0,#2	2
		-------	ADDI R3,R0,#3	3
inicio:	LD R5,0(R10)	-------		4
	SD 0(R20),R1	-------		5
		-------	SUBI R6,R5,#20	6
		-------	BLT R6,else	7
		-------		8
		-------	JMP final	9
else:		-------	SUBI R6,R5,#10	10
	SD 0(R20),R2	-------	BLT R6,else2	11
		-------		12
		-------	JMP final	13
		-------		14
else2:	SD 0(R20),R3	-------		15
final:		-------	SUBI R10,R10,#4	16
		-------	SUBI R20,R20,#4	17
		-------	BNEZ R10,inicio	18
				19

Código VLIW para el apartado (c).

	Operaciones de carga/almacenamiento	Operaciones en coma flotante	Operaciones enteras y saltos	
`inicio:`	`LD R5,0(R10)`			1
			`PRED_CL p3,p4`	2
			`PRED_LT p1,p2,R5,#20`	3
			`PRED_LT p3,p4,R5,#10 (p1)`	4
			`ADDI R1,R0,#1 (p2)`	5
			`ADDI R1,R0,#2 (p4)`	6
			`ADDI R1,R0,#3 (p3)`	7
	`SD 0(R20),R1`		`SUBI R10,R10,#4`	8
			`SUBI R20,R20,#4`	9
			`BNEZ R10, inicio`	10
				11

Código VLIW para el apartado (c) optimizado todo lo posible.

	Operaciones de carga/almacenamiento	Operaciones en coma flotante	Operaciones enteras y saltos	
`inicio:`	`LD R5,0(R10)`		`SUBI R10,R10,#4`	1
			`PRED_CL p3,p4`	2
			`PRED_LT p1,p2,R5,#20`	3
			`PRED_LT p3,p4,R5,#10 (p1)`	4
			`ADDI R1,R0,#1 (p2)`	5
			`ADDI R1,R0,#2 (p4)`	6
			`ADDI R1,R0,#3 (p3)`	7
	`SD 0(R20),R1`		`BNEZ R10,inicio`	8
			`SUBI R20,R20,#4`	9