```
      (JMP ,etat-initial)) ; Saute a l'etat initial
    (mapcar (lambda (etat) ; Genere les instructions pour chaque etat
              (gen-etat-automate etat automate))
            etats)
    `((LABEL accepter) ; Label pour acceptation
      (MOVE R2 R0) ; Indique une acceptation reussie
      (HALT) ; Fin de l'execution

      (LABEL refuser) ; Label pour refus
      (MOVE (:CONST nil) R0) ; Indique un echec
      (HALT))))) ; Fin de l'execution
```

2 Examen de 2021, session 1

Exercice 1 :
Soit la fonction f suivante :

$$f(n) = \begin{cases} n - 10, & \text{si } n > 100 \\ f(f(n + 11)), & \text{sinon} \end{cases}$$

1. Que calcule cette fonction pour $n \leq 101$? Justifier en déroulant son exécution sur au moins deux exemples.
2. Écrire la fonction f en PP.
3. Traduire la fonction f en UPP.
4. Traduire la fonction f en RTL.
5. Traduire la fonction f en ERTL.

1. Calculons des valeurs au hasard :

$$f(101) = 91$$
$$f(100) = f(f(111)) = f(101) = 91$$

   Il semblerait donc que cette fonction calcule toujours 91.

2.
```
function f(n: integer): integer
    if n > 100 then
        f := n - 10
    else
        f := f(f(n + 11));
```

3. Soit donc en UPP :

```
function f(n)
    if n > 100 then
        f := n - 10
    else
        f := f(f(n + 11));
```

4. Soit la traduction en RTL :

```
function f(%0): %1
var %0 %1 %2 %3 %4
entry f6
```

```
exit f0

f6: addiu %2 %0 -100 -> f5
f5: bgtz %2 -> f4, f3

f4: addiu %1 %0 -10 -> f0

f3: addiu %3 %0 11 -> f2
f2: call %4 f(%3) -> f1
f1: call %1 f(%4) -> f0
```

5. Et enfin, en ERTL :

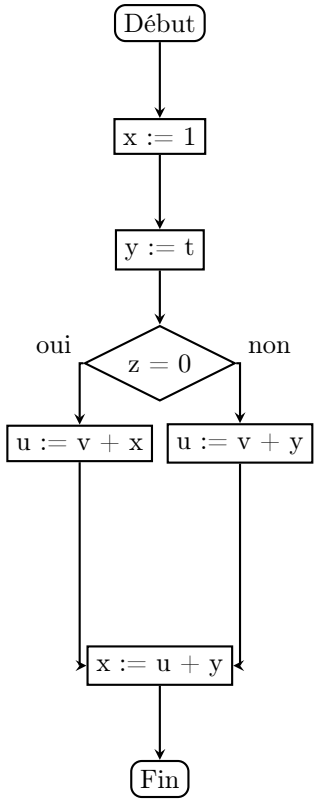
```
procedure f(1)
var %0 %1 %2 %3
f0: newframe -> f1
f1: move %0 $a0 -> f2
f2: move %1 $s0 -> f3
f3: move %2 $s1 -> f4
f4: move %3 $ra -> f5

f5: li $s0 100 -> f6
f6: slt $s1 $a0 $s0 -> f7
f7: bgtz $s1 -> f8, f9

f8: addiu $v0 $a0 -10 -> f14

f9: addiu $s0 $a0 11 -> f10
f10: move $a0 $s0 -> f11
f11: jal f -> f12
f12: move $a0 $v0 -> f13
f13: jal f -> f14

f14: move $a0 %0 -> f15
f15: move $s0 %1 -> f16
f16: move $s1 %2 -> f17
f17: move $ra %3 -> f18
f18: delframe -> f19
f19: jr $ra
exit f19
```

Exercice 2 :
Soit le programme PP suivant :

```
x := 1;
y := t;
if z = 0 then
    u := v + x
else
    u := v + y;
x := u + y
```

1. Dessiner le graphe de flot de contrôle de ce programme.
2. Faire une analyse de durée des variables sachant qu'à la fin du programme, x et y sont vivantes.
3. Dessiner le graphe d'interférences correspondant.
4. Colorier le graphe d'interférences avec 3 couleurs. Doit-on « spiller » x ? Mêmes questions avec 2 couleurs.

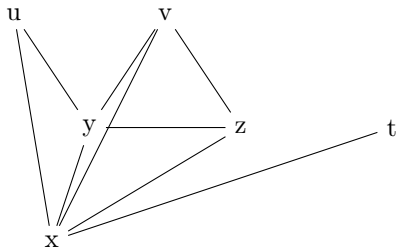
1.



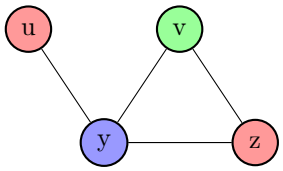
2. Analysons les durées de vie des variables :

```
x := 1; {v, t}
y := t; {v, x, t}
if z = 0 then {v, x, y, z}
    u := v + x {v, x, y}
else
    u := v + y; {v, y}
x := u + y {u, y}
{x, y}
```

3. Soit les interférences : $i = \{(x,y),(u,y),(u,x),(z,x),(z,y),(z,v),(y,v),(y,x),(x,v),(x,t)\}$



4. On ne peut pas colorier le graphe avec 3 couleurs sans spiller (le sommet x possède 4 voisins). Nous allons donc spiller x. Cela nous permet d'aussi éliminer t de la coloration car il devient un sommet isolé. On peut lui assigner le registre que l'on souhaite.



5. Si ici on veut procéder à une 2-coloration, il nous refaut spiller. Par exemple, nous allons spiller v