Langages et Compilation : génération de code

Exercices

Exercice 1.

En utilisant les fonctions de génération de code vues en cours, écrivez la séquence de code produite pour les fragments de programme suivants. On suppose que les déplacements associés à `x1`, `x2` et `x3` sont 4, 8 et 12.

```
x1 := 3 ; x2 := 3 + x1 + x2 ;
si non (x2 = x1) alors
   x1 := x2
sinon
   x2 := 0
```

```
x2 : = 0 ;
tantque ((x3 < 4) and (x2 = x1))
   x3 := x3 + 1;
```

Exercice 2.

On étend le langage `while` en lui ajoutant :
- l'opérateur booléen "ou exclusif" (`e1 xor e2` est vrai ssi un seul des deux opérandes est vrai) ;
- l'instruction `repeter c jusqua e`.

Complétez les fonctions de génération de code vues en cours pour prendre en compte ces extensions.

Exercice 3.

```
void P() {
  int x;

  void P1 (int a) {
    int x1 ;

    void P2(int b, int c) {
      int x2;

      /* code de P2 */
      x2 = c;
      x = x1 + x2 + b;    /* instruction (2) */
    } /* end P2 */

     /* code de P1 */
     x1 =  a;
     P2 (x+1, a);   /* instruction (1) */
     x = 2;
  } /* end P1 */

  /* code de P */
  x = 0;
  P1 (5);
} /* end P */

main () {
  P();
}
```

1. Dessinez le contenu de la pile lors de l'exécution de la procédure `P2`.
2. Donnez le code obtenu pour les instructions **(1)** et **(2)**.

Exercice 4.

```
procedure P(void);
  var z : integer;
  procedure P1(function q(c:integer):integer);
  begin
       z:=q(4);
  end; { P1 }
  function R (x : integer):integer;
  begin
       R:=x+2; // Le résultat de R est la valeur x+2
  end; { R }
  procedure P2(void);
  begin
       P1(R);
  end; { P2 }
begin
  P2();
end;
```

1. Dessinez l'arbre des appels correspondant à l'exécution de cette procédure
2. Dessinez le contenu de la pile lors de l'exécution de `R`
3. Ecrivez le code associé aux procédures `P`, `R`, `P1` et `P2`.