Exercice 1 : somme et produit

```
void addmul1(int*s, int*p,   int a, int b)  { *s=a+b; *p=a*b; }
void addmul2(int*s, int*p) { int a=*s,  b=*p; *s=a+b; *p=a*b; }
           //   r0     r1       r2      r3
addmul2:                        addmul2:
load r0,r2                      24 0 2 *
load r1,r3                      24 1 3 *
addmul1:                        addmul1:
add r2,r3,r2                    0  2 3 2
store r0,r2                     25 0 2 *
sub r2,r3,r2                    2  2 3 2
mul r2,r3,r2                    20 2 3 2
store r1,r2                     25 1 2 *
ret                             29 * * *
```

Ces deux procédures calculent la somme et le produit de deux entiers et rangent les nombres calculés dans des variables pointées par leur deux premiers arguments. Les deux nombres à additionner et multiplier sont les deux derniers arguments d'une des procédures et les anciennes valeurs des variables où mettre les résultats de l'autre procédure.

Expliquez pourquoi il n'y a pas de `ret` après `load r1,r3`.

Enlevez le `sub` sans le remplacer par une autre instruction, mais en utilisant un registre de plus.

Comparez les deux versions. Quelle est la meilleure?

Donnez le contenu du tableau `t` après les instructions :

```
int t[]={1,2,3,4,5,6}; addmul1(t,t+3,2,t[4]); addmul2(t+3,t+5);
```

Exercice 2 :

```
p:
loadimm16 r2,1
add r0,r2,r1
add r1,r2,r2
load r0,r0
load r1,r1
sub r0,r1,r3
movcl r1,r0
add r0,r0,r0
store r2,r0
ret
```

Ecrivez la procédure précédente en binaire.

Donnez le contenu du tableau `t` après l'exécution de :

```
int t[]={1,2,3,4,5,-6,5,4,-3,2,1,0}; p(t); p(t+4); p(t+7); p(t+9);
```

Dites ce que fait la procédure `p`.

Refaites tout l'exercice en remplaçant `movcl` par `movcg`.

Refaites l'exercice en remplaçant `movcl` par `movca`.

Refaites l'exercice en remplaçant `movcl` par `movcb`.

Exercice 3 : compilation d'une formule

Transformez en assembleur, puis en binaire, les instructions suivantes.

On suppose que l'on sait déjà dans quels registres les variables sont rangées et qu'elles ont déjà des valeurs.

Il faut utiliser le moins de registres supplémentaires possible, parmi `r5`, `r6` etc..

On peut utiliser la commutativité et l'associativité de l'addition et la multiplication. On peut aussi détecter les sous-expressions communes pour diminuer le nombres d'opérations. Par exemple `a-b` apparaît dans `a-b+c` et dans `-a+b+c` si on le réécrit `c-(a-b)`.

On utilisera aussi éventuellement la distributivité de la multiplication par rapport à l'addition.

```
//   r0 r1 r2 r3 r4
int  a, b, c, d, e;
a=(a+b)*e+(a-c)*d*a;
b=(a+b+c)*(a+b-c)*(a-b+c)*(-a+b+c);
c=a*b*(a+d*c+e)*d;
d+=a*b*c+a*c*e;
e+=3*a;
```

Exercice 4 :

```
p2:
loadimm16 r2,1
xor r3,r3,r3
l4:
load r0,r4
xor r3,r4,r3
store r0,r3
add r0,r2,r0
sub r1,r2,r1
jnc l4
ret
```

Ecrivez la procédure précédente en binaire.

Que contient le tableau `t` après

```
int t[]={1,-6,2,4,-3,7,5}; p2(t,5);
```

Exercice 5 :

```
p3:
loadimm16 r5,1
l5:
sub r0,r5,r0
jc l6
load r1,r6
load r2,r7
sub r6,r7,r8
movcg r6,r8
movcg r7,r6
movcg r8,r7
store r3,r6
store r4,r7
add r1,r5,r1
add r2,r5,r2
add r3,r5,r3
add r4,r5,r4
jmp l5
l6:
ret
```

Ecrivez la procédure précédente en binaire.

Que contient le tableau `t` après

```
int t[]={1,4,2,-6,-3,7,5,3}; p3(2,t,t+2,t+4,t+6);
```

Que contient le tableau `t` après

```
int t[]={1,4,2,-6,-3,7,5,3}; p3(3,t+5,t+3,t,t+2);
```