## 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            a=*s;                                 24 0 2 *
load r1,r3            b=*p;                                 24 1 3 *
addmul1:                                                    addmul1:
add r2,r3,r2          a+=b;                                 0  2 3 2
store r0,r2           *s=a;                                 25 0 2 *
sub r2,r3,r2          a-=b;                                 2  2 3 2
mul r2,r3,r2          a*=b;                                 20 2 3 2
store r1,r2           *p=a;                                 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.

La procédure `addmul2` range `*s` et `*p` dans les registres `r2` et `r3`, exactement là où doivent se trouver les troisième et quatrième arguments d'une procédure, puis elle exécute `addmul1` comme si on avait écrit :

```
void addmul2(int*s, int*p) { addmul1(s,p,*s,*p); }
```

Mais l'appel à `addmul1` suivi d'un `return;` est remplacé par un branchement sur le début de `addmul1`. C'est pourquoi il n'y a pas de `ret` après les deux `load`.

Pour économiser un registre, la somme de `a` et `b` est mise à la place de `a`. On est donc obligé de recalculer la valeur initiale de `a` par `a-=b;` avant de faire le produit par `b`. Si on utilise un registre supplémentaire `r4` pour mettre `a+b`, on économise une instruction:

```
void addmul1(int*s, int*p,   int a, int b)  { *s=a+b; *p=a*b; }
          //   r0     r1       r2      r3       r4
addmul1:                                                    addmul1:
add r2,r3,r4          c=a+b;                                0  2 3 4
store r0,r4           *s=c;                                 25 0 4 *
mul r2,r3,r2          a*=b;                                 20 2 3 2
store r1,r2           *p=a;                                 25 1 2 *
ret                                                         29 * * *
```

Une des versions utilise un registre de plus, alors que l'autre a une instruction de plus.

```
int t[]={1,2,3, 4,5,6 };  addmul1(t,t+3,2,t[4]); // a=2, b=t[4]=5,
      //                  *t=t[0]=a+b=2+5=7, *(t+3)=t[3]=a*b=2*5=10
      /*{7,2,3,10,5,6 }*/ addmul2(t+3,t+5);      // a=t[3]=10, b=t[5]=6,
      //                  t[3]=10+6=16,  t[5]=10*6=60
      /*{7,2,3,16,5,60}*/
```

**Exercice 2 :**

```
p:                              void p(int t[]) // r0    r1    r2   r3
loadimm16 r2,1    26 2 0 1      {               // t           1
add r0,r2,r1      0  0 2 1                      //      t+1
add r1,r2,r2      0  1 2 2                      //            t+2
load r0,r0        24 0 0 *        int a=*t,     // a=*t
load r1,r1        24 1 1 *            b=t[1],   //      b=t[1]
sub r0,r1,r3      2  0 1 3            c=a-b;    //                  c=*t-t[1]
movcl r1,r0       57 1 0 *        if(a<b) a=b;  // max(*t,t[1])
add r0,r0,r0      0  0 0 0        a+=a;         // 2*max(*t,t[1])
store r2,r0       25 2 0 *        t[2]=a;
ret               29 * * *      }
void p(int t[]) { t[2]=2*max(*t,t[1]); }
int t[]={1,2,3,4,5,-6, 5,4,-3,2,1,0};  p(t);   // *t=1   t[1]=2  max=2 t[2]=4
      /*{1,2,4,4,5,-6, 5,4,-3,2,1,0}*/ p(t+4); // t[4]=5 t[5]=-6 max=5 t[6]=10
      /*{1,2,4,4,5,-6,10,4,-3,2,1,0}*/ p(t+7); // t[7]=4 t[8]=-3 max=4 t[9]=8
      /*{1,2,4,4,5,-6,10,4,-3,8,1,0}*/ p(t+9); // t[9]=8 t[10]=1 max=8 t[11]=16
      /*{1,2,4,4,5,-6,10,4,-3,8,1,16}*/
```

Si on remplace `movcl` par `movcg`, alors `if(a<b)` devient `if(a>b)`. On a donc

```
void p(int t[]) { t[2]=2*min(*t,t[1]); }
int t[]={1,2,3,4,5,-6,  5,4,-3, 2,1,0}; p(t);  // *t=1   t[1]=2  min=1  t[2]=2
      /*{1,2,2,4,5,-6,  5,4,-3, 2,1,0}*/p(t+4);// t[4]=5 t[5]=-6 min=-6 t[6]=-12
      /*{1,2,2,4,5,-6,-12,4,-3, 2,1,0}*/p(t+7);// t[7]=4 t[8]=-3 min=-3 t[9]=-6
      /*{1,2,2,4,5,-6,-12,4,-3,-6,1,0}*/p(t+9);//t[9]=-6 t[10]=1 min=-6 t[11]=-12
      /*{1,2,2,4,5,-6,-12,4,-3,-6,1,-12}*/
```

En remplaçant `movcl` par `movca`, on calcule toujours le double du maximum, mais les nombres comparés sont non signés. En fait tout se passe comme si les nombres négatifs étaient plus grands que les positifs.

```
void p(unsigned t[]) { t[2]=2*max(*t,t[1]); }
int t[]={1,2,3,4,5,-6,  5,4,-3, 2,1,0}; p(t);  // *t=1   t[1]=2  max=2  t[2]=4
      /*{1,2,4,4,5,-6,  5,4,-3, 2,1,0}*/p(t+4);// t[4]=5 t[5]=-6 max=-6 t[6]=-12
      /*{1,2,4,4,5,-6,-12,4,-3, 2,1,0}*/p(t+7);// t[7]=4 t[8]=-3 max=-3 t[9]=-6
      /*{1,2,4,4,5,-6,-12,4,-3,-6,1,0}*/p(t+9);//t[9]=-6 t[10]=1 max=-6 t[11]=-12
      /*{1,2,4,4,5,-6,-12,4,-3,-6,1,-12}*/
```

En remplaçant `movcl` par `movcb` on obtient:

```
void p(unsigned t[]) { t[2]=2*min(*t,t[1]); }
int t[]={1,2,3,4,5,-6, 5,4,-3,2,1,0};   p(t);   // *t=1   t[1]=2  min=1 t[2]=2
      /*{1,2,2,4,5,-6, 5,4,-3,2,1,0}*/  p(t+4); // t[4]=5 t[5]=-6 min=5 t[6]=10
      /*{1,2,2,4,5,-6,10,4,-3,2,1,0}*/  p(t+7); // t[7]=4 t[8]=-3 min=4 t[9]=8
      /*{1,2,2,4,5,-6,10,4,-3,8,1,0}*/  p(t+9); // t[9]=8 t[10]=1 min=1 t[11]=2
      /*{1,2,2,4,5,-6,10,4,-3,8,1,2}*/
```

### 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   r5      r6
                          int  a, b, c, d, e;
sub r0,r2,r5    2  0 2 5  //                    a-c
mul r5,r3,r5    20 5 3 5  //                    (a-c)*d
mul r5,r0,r5    20 5 0 5  //                    (a-c)*d*a
add r0,r1,r0    0  0 1 0  //   a+b
mul r0,r4,r0    20 0 4 0  //   (a+b)*e
add r0,r5,r0    0  0 5 0       a=(a+b)*e+(a-c)*d*a;
sub r0,r1,r5    2  0 1 5  //                    a-b
add r0,r1,r1    0  0 1 1  //      a+b
add r1,r2,r6    0  1 2 6  //                           a+b+c
sub r1,r2,r1    2  1 2 1  //      a+b-c
mul r1,r6,r1    20 1 6 1  //      (a+b+c)*(a+b-b)
sub r2,r5,r6    2  2 5 6  //                          -a+b+c
mul r1,r6,r1    20 1 6 1  //      (a+b+c)*(a+b-c)*(-a+b+c)
add r2,r5,r6    0  2 5 6  //                           a-b+c
mul r1,r6,r1    20 1 6 1         b=(a+b+c)*(a+b-c)*(a-b+c)*(-a+b+c);
mul r3,r2,r2    20 3 2 2  //         d*c
add r0,r2,r2    0  0 2 2  //         a+d*c
add r2,r4,r2    0  2 4 2  //         a+d*c+d
mul r1,r2,r2    20 1 2 2  //         b*(a+d*c+d)
mul r0,r2,r2    20 0 2 2  //         a*b*(a+d*c+d)
mul r2,r3,r2    20 2 3 2            c=a*b*(a+d*c+d)*d;
add r1,r4,r5    0  1 4 5  //                    b+e
mul r0,r5,r5    20 0 5 5  //                    a*(b+e)
mul r5,r2,r5    20 5 2 5  //                    a*(b+e)*c
add r3,r5,r3    0  3 5 3               d+=a*b*c+a*c*e;
add r4,r0,r4    0  4 0 4  //               e+a
add r4,r0,r4    0  4 0 4  //               e+2*a
add r4,r0,r4    0  4 0 4                  e+=3*a;
```

**Exercice** 4 **:**

```
p2:                                  void p2(int*t, int n)// 1   s   x
loadimm16 r2,1   26 2 0 1            {      //  r0     r1    r2  r3  r4
xor r3,r3,r3     8  3 3 3              int s=0;
l4:                                    do
load r0,r4       24 0 4 *              { int x=*t;
xor r3,r4,r3     8  3 4 3                s^=x;
store r0,r3      25 0 3 *                *t=s;
add r0,r2,r0     0  0 2 0                t++;
sub r1,r2,r1     2  1 2 1              } while(n--);
jnc l4           33 255 255 250
ret              29 * * *            }
void p2(int *t, int n) { int s=0; do s^=*t, *t++=s; while(n--); }
int t[]={1,-6, 2, 4,-3,7,5}; p2(t,5);
      /*{1,-5,-7,-3, 0,7,5}*/
```

car $0 \oplus 1 = 1$,
$1 \oplus -6 = 1 \oplus 5 \oplus -1 = 4 \oplus -1 = -5$,
$-5 \oplus 2 = -1 \oplus 4 \oplus 2 = -1 \oplus 6 = -7$,
$-7 \oplus 4 = -1 \oplus 6 \oplus 4 = -1 \oplus 2 = -3$,
$-3 \oplus -3 = 0$ et
$0 \oplus 7 = 7$.
La boucle est faite $n + 1{=}5 + 1{=}6$ fois.

**Exercice 5 :**

```
p3:                          void p3(int n,int*t,int*u,int*v,int*w)// 1  x  y  d
loadimm16 r5,1  26 5 0 1     //         r0    r1    r2    r3    r4   r5 r6 r7 r8
l5:
sub r0,r5,r0    2  0 5 0     { while(n--)
jc l6           32 0 0 13      {
load r1,r6      24 1 6 *         int x=*t,
load r2,r7      24 2 7 *             y=*u,
sub r6,r7,r8    2  6 7 8             d=x-y;
movcg r6,r8     54 6 8 *         if(x>y) d=x,  // On échange x et y
movcg r7,r6     54 7 6 *                 x=y,
movcg r8,r7     54 8 7 *                 y=d;
store r3,r6     25 3 6 *         *v=x;
store r4,r7     25 4 7 *         *w=y;
add r1,r5,r1    0  1 5 1         t++;
add r2,r5,r2    0  2 5 2         u++;
add r3,r5,r3    0  3 5 3         v++;
add r4,r5,r4    0  4 5 4         w++;
jmp l5          44 255 255 241  }
l6:
ret             29 * * *     }
```

```
void p3(int n,int*t,int*u,int*v,int*w)
{ while(n--)
  { int x=*t++, y=*u++;
    if(x>y) { int d=x; x=y; y=d; } // on échange x et y
    *v++=x; *w++=y;
  }
}
void p3(int n,int t[n],int u[n],int v[n],int w[n])
{ while(n--)
  { int x=*t++, y=*u++; *v++=min(x,y); *w++=max(x,y); }
}
int t[]={1,4,2,-6,-3, 7,5,3}; p3(2,t,t+2,t+4,t+6);
      //{1,4,2,-6, 1, 7,2,3}  x=*t  =1  y=t[2]= 2  t[4]=min= 1  t[6]=max=2
      //{1,4,2,-6, 1,-6,2,4}  x=t[1]=4  y=t[3]=-6  t[5]=min=-6  t[7]=max=4
int t[]={ 1, 4,2,-6,-3,7,5,3}; p3(3,t+5,t+3,t,t+2);
      //{-6, 4,7,-6,-3,7,5,3}  x=t[5]=7  y=t[3]=-6  *t  =min=-6  t[2]=max=7
      //{-6,-3,7, 5,-3,7,5,3}  x=t[6]=5  y=t[4]=-3  t[1]=min=-3  t[3]=max=5
      //{-6,-3,3, 5, 7,7,5,3}  x=t[7]=3  y=t[5]= 7  t[2]=min= 3  t[4]=max=7
```