Université de Nanterre mardi 13 mai 2014
architecture des ordinateurs examen

```
      p1                p2                  f
  loadimm16 r3,1   loadimm16 r3,1      and       r1,r1,r1
  sub r0,r3,r0     add r0,r1,r0        je        l10
  jc L2            sub r0,r3,r0    l9:
L1:                sub r1,r0,r4        mov       r0,r2
  load r1,r4       ja L4               sarimm    r0,r0,31
  load r2,r5     L3:                   div2      r2,r0,r1
  add  r4,r5,r6    load r1,r4          mov       r1,r0
  mul  r4,r5,r4    load r0,r5          and       r2,r2,r1
  store r1,r6      add r4,r5,r4        jne       l9
  store r2,r4      add r4,r5,r5    l10:
  add r1,r3,r1     store r1,r5         ret
  add r2,r3,r2     store r0,r4
  sub r0,r3,r0     add r1,r3,r1
  jnc L1           sub r0,r3,r0
L2:                sub r1,r0,r4
  ret              jna L3
                 L4:
                   ret
```

Retrouvez le source C des deux procédures `p1` et `p2` et de la fonction `f`.

Modifiez la fonction `f` pour qu'elle fasse des calculs sur des entiers non signés.

Ecrivez l'équivalent en assembleur de la fonction C ayant pour prototype `long som9(int n, int t[]);` qui calcule $\sum_{i=0}^{n-1} t[i]^9$.

Dans une machine qui fait des calculs sur des entiers codés sur 4 bits, quelles valeurs auront les indicateurs CF (carry), OF (overflow), ZF (zéro) et (SF) (signe) après chacune des opérations: 9+10, 9-10, 7+3, 7-3, 13+3, 13-3, 5+13, 5-13, 4+11, 4-11?

Dessinez l'arbre syntaxique de l'instruction `a=(a+b)*(a+c)/(b+c)-a;`

Compilez la en supposant que les variables `a`, `b` et `c` sont rangées dans les registres `r10`, `r11` et `r12` et que vous pouvez modifier à votre guise le contenu des registres `r20` à `r30`.

Refaîtes la même question en supposant les variables `a`, `b` et `c` en mémoire aux adresses `r128+10`, `r128+11` et `r128+12`.

Corrigé

```
void p1(int n, int *t, int *u)
{ while(n--) { int x=*t, y=*u; *t++=x+y, *u++=x*y; } }
void p2(int n, int *t)
{ int *u=t+n-1, x, y;
  while(u>t) x=*t, y+=x+=y=*u, *t++=y, *u--=x;
}
void p2(int n, int *t)
{ int *u=t+n-1, x, y;
  while(u>t) x=*t, y=*u, *t++=x+2*y, *u--=x+y;
}
int f(int a, int b)
{ while(b) { int c=a%b; a=b, b=c; }
  return a;
}
```

`p1(n,t,u)` prend deux tableaux d'entiers de même dimension `n` et remplace le contenu du premier par leur somme, et le contenu du second par leur produit.

`p2(n,t)` prend un tableau d'entiers de dimension `n` et remplace le premier élément du tableau `t[0]` et le dernier `t[n-1]` par `t[0]+2*t[n-1]` et `t[0]+t[n-1]` puis recommence sur le sous-tableau obtenu en ignorant le premier et le dernier élément. On fait cela tant qu'il y a plusieurs éléments dans le tableau. Si le tableau est de taille impaire, son élément central sera inchangé.

`f(a,b)` est le pgcd de `a` et `b`.

Pour que `f` opère sur des entiers non signés, il faut remplacer

```
sarimm    r0,r0,31
div2      r2,r0,r1
```

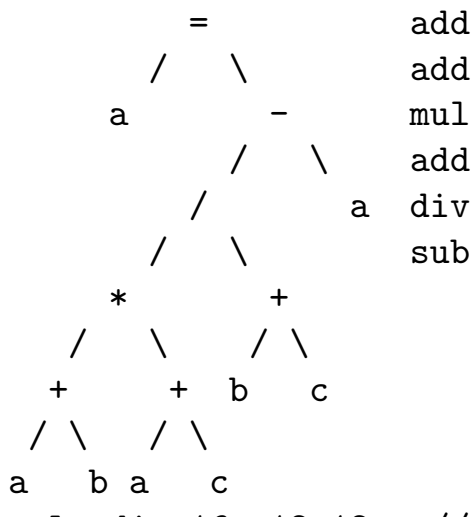
par

```
xor       r0,r0,r0
idiv2     r2,r0,r1
```

La partie haute du dividende n'est plus -1 ou 0 selon le signe de la partie basse, mais toujours 0. On fait une division non signée au lieu d'une division signée.

```
                    // int som9(int n, int *t)
  loadimm16 r2,1    // {int r2=1;
  xor r3,r3,r3      //  int s=0;
  sub r0,r2,r0      //  if(n--)
  jc L2             //  do
L1:                 //  {
  load r1,r4        //    int x=*t;
  mul r4,r4,r5      //    int y=x*x;  // *t^2
  mul r4,r5,r4      //    x*=y;       // *t^3
  mul r4,r4,r5      //    y=x*x;      // *t^6
  mul r4,r5,r4      //    x*=y;       // *t^9
  add r4,r3,r3      //    s+=x;
  add r1,r2,r1      //    t++;
  sub r0,r2,r0      //  } while(n--);
  jnc L1
L2:
  mov r3,r0         //  return s;
  ret               // }
```

non signé	CF	signé	OF	ZF	SF
9+10=3	1	-7+-6=3	1	0	0
9-10=15	1	-7--6=-1	0	0	1
7+3=10	0	7+3=-6	1	0	1
7-3=4	0	7-3=4	0	0	0
13+3=0	1	-3+3=0	0	1	0
13-3=10	0	-3-3=-6	0	0	1
5+13=2	1	5+-3=2	0	0	0
5-13=8	1	5--3=-8	1	0	1
4+11=15	0	4+-5=-1	0	0	1
4-11=9	1	4--5=-7	1	0	1




```
  loadimm16 r12,12   // 12
  loadimm16 r22,1    // 1
  add  r128,r12,r12  // r128+12=&c
  sub  r12,r22,r11   // (r128+12)-1=r128+11=&b
  sub  r11,r22,r22   // (r128+11)-1=r128+10=&a
  load r12,r12       // c
  load r11,r11       // b
  load r22,r10       // a
  add  r10,r11,r20   // a+b
  add  r10,r12,r21   // a+c
  mul  r20,r21,r20   // (a+b)*(a+c)
  add  r11,r12,r21   // b+c
  div  r20,r21,r20   // (a+b)*(a+c)/(b+c)
  sub  r20,r10,r10   // (a+b)*(a+c)/(b+c)-a
  store r22,r10      // a
```

Barème

p1	3pt	2.5pt pour du vrai C ou 1pt si r4=*r1 ... 0.5pt pour une explication en français
p2	3pt	idem
f	3pt	idem
f non signée	1pt	0.5pt pour `sarimm r0,r0,31` → `xor r0,r0,r0` 0.5pt pour `div2 r2,r0,r1` → `idiv2 r2,r0,r1`
$\sum_{i=0}^{n-1} t[i]^9$	3pt	1pt boucle 0.5pt valeur de retour 1.5pt x^9 en 4 instructions ou 1pt si juste mais plus de 4 instr. ou 0.5pt si idée
9+10	7.5pt	=10x0.75pt Chaque opération est notée comme un QCM indépendant avec une note entre 0 et 0.75pt. CF juste:0.25 absent:0 faux:-0.25 OF:0.25, ZF:0.125, SF:0.125
arbre	1pt	
a=r10	3pt	-0.5pt par opération fausse
a=*(r128+10)	2pt	0.5pt `load` 0.5pt `load` 0.5pt `load` 0.5pt `store` 26.5pt