```
 0, 1, 3, *        load r1,r3          r3=*r1
 1, 1, 3, *        store r1,r3         *r1=r3
 2, 2, 5, 4        loadimm16  r2,1284  r2=5*256+4
 3, *, *, *, 34567 loadimm32  r3,34567 r3=34567
 4, 7, 5, 3        add r7,r5,r3        r3=r7+r5
 5, 3, 4, 5        adc r3,r4,r5        r5=r3+r4+cf
 6, 9, 3, 4        sub r9,r3,r4        r4=r9-r3
 7, 8, 1, 0        sbb r8,r1,r0        r0=r8-r1-cf
 8, 4, 6, *        com r4,r6           r6=~r4
 9, 5, 7, *        neg r5,r7           r7=-r5
10,                jc = jb = jnae
12,                je = jz
14,                ja = jnbe
16,                jg
18,                jgz
20,                jo
22,                jmp
23, *, *, *        nop                 ne fait rien
24, 4, 5, 7        or r4,r5,r7         r7=r5|r4
25, 5, 3, 8        and r5,r3,r8        r8=r5&r3
26, 3, 6, 4        xor r3,r6,r4        r4=r3^r6
27, 5, 3, 4        shr r5,r3,r4        r4=r5>>r3  non signé
28, 7, 4, 3        shrimm r7,4,r3      r3=r7>>3
29, 5, 6, 9        shl r5,r6,r9        r9=r5<<r6
30                 shlimm
31, 6, 5, 8        sar r6,r5,r8        r8=r6>>r5  signé
32                 sarimm
34, 1, 5, 4        div r1,r5,r4        r4=r1/r5  signé
35,                idiv                          non signé
36, 8, 4, 2        div2 r8,r4,r2       (r4<<32+r8) divisé par r2  signé
37, 8, 4, 2        idiv2 r8,r4,r2      r8=reste r4=quotient      non signé
38, 1, 5, 8        mul r1,r5,r8        r8=r1*r5
40, 2, 6, 9        mul2 r2,r6,r9       (r6<<32+r9)=r2*r6  signé
41,                imul2                                  non signé
```

```
void echtab(int n, int *t, int *u)
{ while(n--) { int x=*t, y=*u; *t++=y, *u++=x; } }
echtab:  //  r0  r1  r2  r3  r4  r5
         //   n   t   u   1   x   y
  loadimm16  r3,1       // r3=1
  sub        r0,r3,r0   // if(!n--) goto l2
  jc l2
l1:
  load       r1,r4      // x=*t
  load       r2,r5      // y=*u
  store      r2,r4      // *u=x
  store      r1,r5      // *t=y
  add        r1,r3,r1   // ++t
  add        r2,r3,r2   // ++u
  sub        r0,r3,r0   // if(n--) goto l1
  jnc l1
l2:
  ret
void rettab(int n, int *t)
{ int *u=t+n-1; while(u>t) { int x=*t, y=*u; *t--=y, *u++=x; } }
rettab:  //  r0  r1  r2  r3  r4  r5
         //   n   t   u   1   x   y
  loadimm16  r3,1       // r3=1
  add        r1,r0,r2   // u=t+n
  sub        r2,r3,r2   // u=t+n-1
  sub        r1,r2,r4   // if(t>=u) goto l4
  jnc l4
l3:
  load       r1,r4      // x=*t
  load       r2,r5      // y=*u
  store      r2,r4      // *u=x
  store      r1,r5      // *t=y
  add        r1,r3,r1   // ++t
  sub        r2,r3,r2   // --u
  sub        r1,r2,r4   // if(t<u) goto l3
  jc l3
l4:
  ret
```

Dans la procédure précédente, on peut économiser un registre en rangeant `u` et `n` dans le même registre. On peut donc remplacer toutes les occurences de `r2` par `r0`.

```
int somt4(int n, int *t) // ∑_{i=0}^{n-1} t[i]^4
{ int s=0, x; while(n--) x=*++t, x*=x, s+=x*x; return s; }
sont4:  //  r0  r1  r2  r3  r4
        //   n   t   1   s   x
  loadimm16  r2,1       // r2=1
  xor        r3,r3,r3   // s=0
  sub        r0,r2,r0   // if(!n--) goto l6
  jc L6
L5:
  load       r1,r4      // x=*t
  mul        r4,r4,r4   // x*=x
  mul        r4,r4,r4   // x*=x
  add        r3,r4,r3   // s+=x
  add        r1,r2,r1   // t++
  sub        r0,r2,r0   // if(n--) goto l5
  jnc L5
L6:
  mov r3,r0             // return s
  ret
int pgcd(int a, int b) { int c; while(b) c=a%b, a=b, b=c; return a; }
pgcd: //  r0  r1  r2
      //   a   b   c
   and       r1,r1,r1
   je        l10
l9:
   mov       r0,r2      // c=a
   sarimm    r0,r0,31   // r0,r2=a (extension du signe)
   div2      r2,r0,r1   // c=a%b, a=a/b
   mov       r1,r0      // a=b
   and       r2,r2,r1   // b=c
   jne       l9
l10:
   ret
```

```
void mul224(int a[2], int b[2], int c[4]); // c[0..3]=a[0..1]*b[0..1]
// r0  r1  r2  r3   r4   r5   r6   r7   r8  r9  r10  r11  r12 r13 r14
//  a   b   c   1 a[0] a[1] b[0] b[1] c[0] c[1] c[2] c[3]   l   h   0
mul224:
  xor        r14,r14,r14// r14=0
  loadimm16  r3,1       // r3=1
  load       r0,r4      // a[0]
  add        r0,r3,r0   // a+1
  load       r0,r5      // a[1]
  load       r1,r6      // b[0]
  add        r1,r3,r1   // b+1
  load       r1,r7      // b[1]
  mov        r4,r9      // a[0]
  mul2       r6,r9,r8   // r9,r8=a[0]*b[0]
  mov        r5,r11     // a[1]
  mul2       r7,r11,r10 // r11,r10=a[1]*b[1]
  mov        r4,r13     // a[0]
  mul2       r7,r13,r12 // r13,r12=a[0]*b[1]
  add        r9,r12,r9
  adc        r10,r13,r10  // r11,r10,r9+=r14,r13,r12
  adc        r11,r14,r11
  mov        r5,r13     // a[1]
  mul2       r6,r13,r12 // r13,r12=a[1]*b[0]
  add        r9,r12,r9
  adc        r10,r13,r10  // r11,r10,r9+=r14,r13,r12
  adc        r11,r14,r11
  and        r5,r5,r5
  jge l7                // if(a[1]<0) c[3],c[2]-=b[1],b[0]
  sub        r10,r6,r10
  sbb        r11,r7,r11
l7:
  and        r7,r7,r7
  jge l8                // if(b[1]<0) c[3],c[2]-=a[1],a[0]
  sub        r10,r4,r10
  sbb        r11,r5,r11
l8:
  store      r2,r8      // c[0]
  add        r2,r3,r2   // c+1
  store      r2,r9      // c[1]
  add        r2,r3,r2   // c+2
  store      r2,r10     // c[2]
  add        r2,r3,r2   // c+3
  store      r2,r11     // c[3]
  ret
```

En supposant maintenant que les arguments sont passés dans la pile, on va compiler le programme suivant sans optimiser en mettant les variables locales dans la pile :

```
int sqr(int a) { return a*a; }
int somt2(int n, int t[])
{ int s=0, i;
  for(i=0;i<n;i++) s+=sqr(t[i]);
  return s;
}
sqr:   // r0      *r128     r128[1]
       // a*a     &retour     a
  loadimm16  r0,1       // 1
  add        r128,r0,r0 // &a
  load       r0,r0      // a
  mul        r0,r0,r0   // a*a
  ret                   // return a*a
somt2:          //    *r128     r128[1]  r128[2]
// r128[0]  r128[1]  r128[2]    r128[3]  r128[4]
//    s        i     &retour       n        t
  loadimm16  r0,2       // 2     allocation de mémoire dans la pile
  sub        r128,r0,r128// r128-=2        pour s et i
  xor        r0,r0,r0   // 0
  store      r128,r0    // s=0
  xor        r0,r0,r0   // 0
  loadimm16  r0,1       // 1
  add        r128,r0,r1 // &i
  store      r128,r0    // i=0
l11:
  loadimm16  r0,1       // 1
  add        r0,r128,r0 // &i
  load       r0,r0      // i
  loadimm16  r1,3       // 3
  add        r1,r128,r1 // &n
  load       r1,r1      // n
  sub        r0,r1,r1   // i-n
  jge l12               // if(i>=n) goto l12
  loadimm16  r0,1       // 1
  add        r0,r128,r0 // &i
  load       r0,r0      // i
  loadimm16  r1,4       // 4
  add        r1,r128,r1 // &t
  load       r1,r1      // t
  add        r0,r1,r0   // t+i
  load       r0,r0      // t[i]
  loadimm16  r1,1       // 1
```

```
  sub         r128,r1,r118// r128--
  store       r128,r0     // on empile t[i]
  call        sqr         // r0=sqr(t[i])
  loadimm16   r1,1        // 1
  add         r128,r1,r118// r128++
  load        r128,r1     // s
  add         r1,r0,r1    // s+sqr(t[i])
  store       r128,r1     // s+=sqr(t[i])
  loadimm16   r0,1        // 1
  add         r0,r128,r0  // &i
  load        r0,r1       // i
  loadimm16   r2,1        // 1
  add         r2,r1,r1    // i+1
  store       r0,r1       // i++
  jmp l11
l12:
  load        r128,r1     // s
  loadimm16   r0,2        // 2     libération de la mémoire dans la pile
  add         r128,r0,r128// r128+=2     pour s et i
  ret                     // return s
```

On peut faire diverses optimistions dans ce code : On peut remplacer l'appel à `sqr` par son code, cela économise toutes les manipulations de pile. On peut mettre les variables `n`, `t`, `s` et `i` dans les registres au lieu de les mettre en mémoire dans la pile. Enfin on peut utiliser un registre qui contient toujours 1, cela évite les `loadimm16` dans la boucle :

```
somt2: // r0  r1  r2  r3  r4   r5      *r128     r128[1]  r128[2]
       //  s   1   n   t   i    *      &retour     n        t
  xor         r0,r0,r0    // s=0
  loadimm16   r1,1        // r1=1
  add         r128,r1,r2  // &n
  add         r2,r1,r3    // &t
  load        r2,r2       // n
  load        r3,r3       // t
  xor         r4,r4,r4    // i=0
  sub         r4,r2,r5    // if(i>=n) goto l10
  jge l10
l9:
  add         r3,r4,r5    // t+i
  load        r5,r5       // t[i]
  mul         r5,r5,r5    // sqr(t[i])
  add         r0,r5,r0    // s+=sqr(t[i])
  add         r4,r1,r4    // i++
  sub         r4,r2,r5    // if(i<n) goto l9
  jl l9
l10:   ret                // return s
```