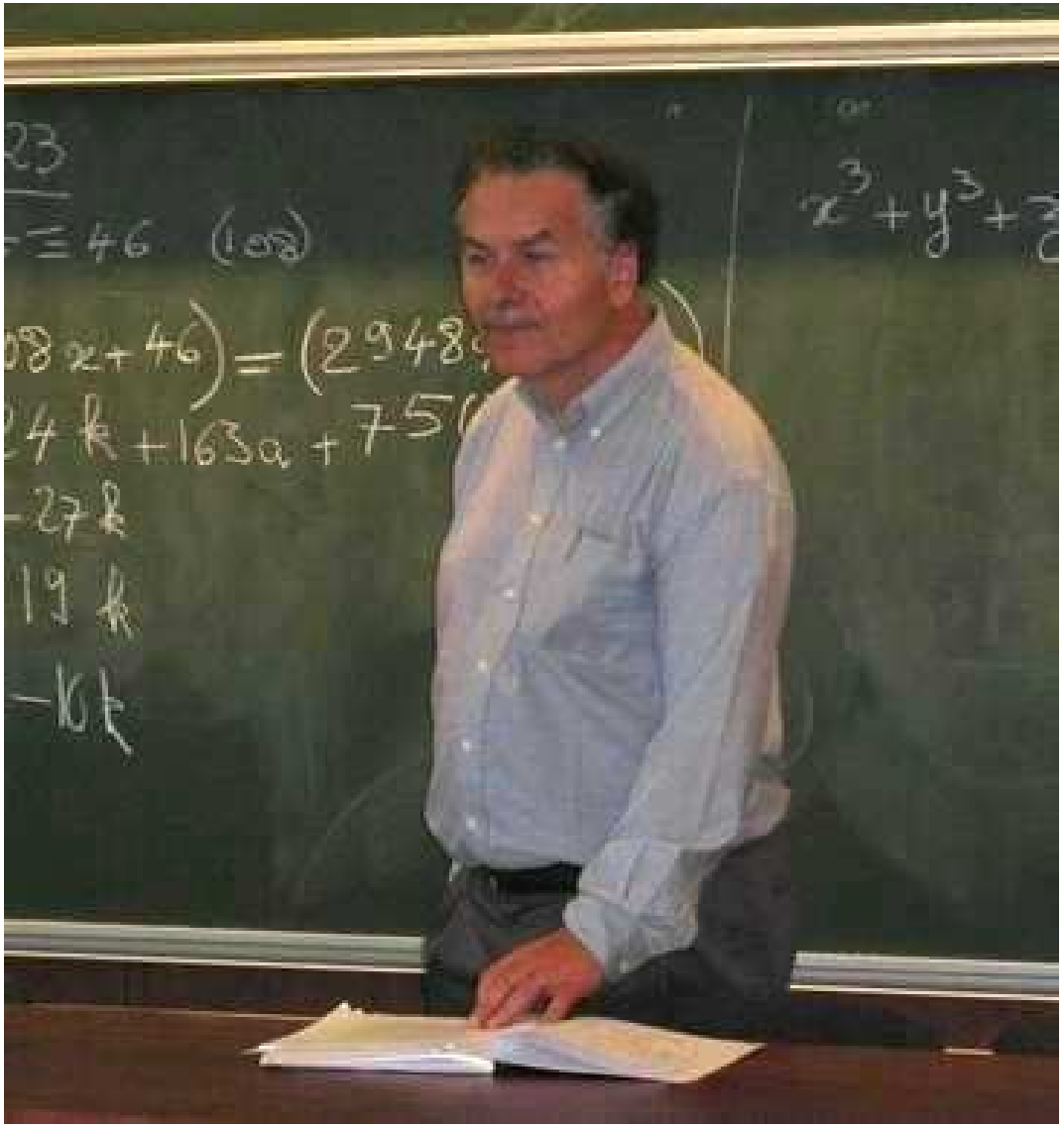
The Ups and Downs of PARI/GP in the last 20 years

Paul Zimmermann

(with many thanks to Karim Belabas)

Explicit Methods in Number Theory, October 15th-19th 2007, Bordeaux

Henri as a Mathematician



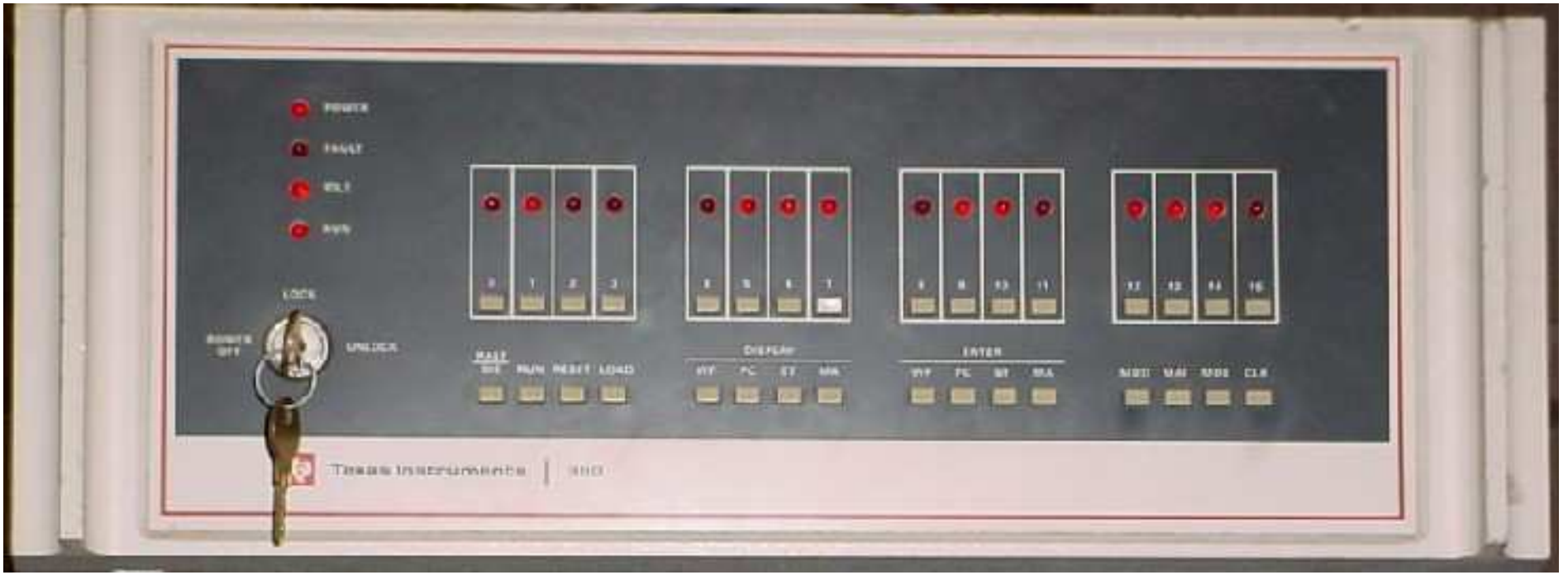
Henri as a Player

Henri as a Hacker

Il était une fois . . .

1975: CNRS *Action Thématique Programmée* “Conception et réalisation de langages ou de systèmes de manipulation formelle adaptés à des branches de mathématiques” ‡

$\Longrightarrow$ a TI 980 machine in Bordeaux (assembly, Fortran)



(here a TI 990)

ISABELLE

Henri and François Dress

First result: $2 + 2$ gives . . .

ISABELLE

Henri and François Dress

First result: $2 + 2$ gives . . . 6813!

ISABELLE

Henri and François Dress

First result: $2 + 2$ gives . . . 6813!

Real birth on 20 July 1979

```
//EX,,#ISAB

BONJOUR! JE M'APPELLE ISABELLE ET JE SUIS PRETE A TOUT
DEPUIS LE 20 JUILLET 1979
PROG = 05004, DOUB = 00100, MULT = 01335

!2+2
           4

!P
PRECISION  EST   00035 DECIMALES

!1000P
PRECISION ETAIT  00035 DECIMALES
PROG = 05175, DOUB = 00100, MULT = 00038

!1000W
```

```
!PI
  3.14159265358979323846264338327950288419716939937510582097494459230
78164062862089986280348253421170679821480865132823066470938446095505822
23172535940812848111745028410270193852110555964462294895493038196442881
09756659334461284756482337867831652712019091456485669234603486104543266
48213393607260249141273724587006606315588174881520920962829254091715364
36789259036001133053054882046652138414695194151160943305727036575959195
30921861173819326117931051185480744623799627495673518857527248912279381
83011949129833673362440656643086021394946395224737190702179860943702770
53921717629317675238467481846766940513200056812714526356082778577134275
77896091736371787214684409012249534301465495853710507922796892589235420
19956112129021960864034418159813629774771309960518707211349999983729780
49951059731732816096318595024459455346908302642522308253344685035261931
18817101000313783875288658753320838142061717669147303598253490428755468
73115956286388235378759375195778185778053217122680661300192787661119590
9216420198
```

“On abandonne maintenant ces petits jeux et on passe à la réalisation et à l’exécution de programmes :”

“On abandonne maintenant ces petits jeux et on passe à la réalisation et à l’exécution de programmes :”

```
L00010:D9=K
L00020:D0=0
L00030:D1=D2=1
L00040:D0=D0+(MOBD1)*D9/D2
L00050:GTL40*PZD9-D2=D1*D1=D1+1
L00060:K=(INVD9^.4)*K=(K=D0)-D9*6:PI*PI
L00070:GTL_1
```

Computes $Q(x) = \sum_{n \leq x} |\mu(n)|$ with $Q(x) = \sum_{a \leq \sqrt{x}} \mu(a) \lfloor \frac{x}{a^2} \rfloor$.

```
!GSL10

?100
      61
?1000
      608
?10000
      6083
?100000
      60794
?1000000
      607926
?10000000
      6079291
```

Time: 85.5s for 10^7.

New Computer

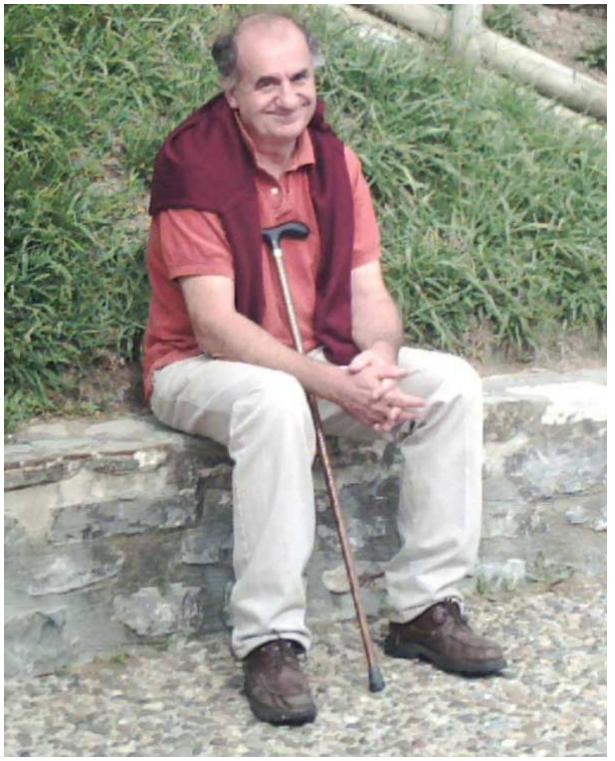
SORD M68

68000 assembly (16/32 bits)

Pascal language

DDT debugger

New People



First success

One year work writing the 68000 assembly kernel (Batut, Cohen, Olivier)

First success

One year work writing the 68000 assembly kernel (Batut, Cohen, Olivier)

30000 digits of π

First success

One year work writing the 68000 assembly kernel (Batut, Cohen, Olivier)

30000 digits of π

Check $\exp(\log(2)) = 2$ to arbitrary precision.

First success

One year work writing the 68000 assembly kernel (Batut, Cohen, Olivier)

30000 digits of π

Check $\exp(\log(2)) = 2$ to arbitrary precision.

Start to implement in Pascal the generic operations

$$\text{PARI} = \text{PASCAL} + \text{ARITHMÉTIQUE}$$

Sun 3/260

True 32-bit: full rewrite of assembly kernel (6000 lines, 6 months)!

Switch to C for generic operations and transcendental functions.

68020 assembly kernel: floating-point addition

```
#=============================================================#
#                                                             #
#               Addition : reel + reel = reel                 #
#                                                             #
#       entree : a7@(4) pointe sur r2 de type R               #
#                a7@(8) pointe sur r1 de type R               #
#       sortie : d0 pointe sur r2 + r1 de type R (zone creee) #
#       precision : L = inf ( L2 , L1 + [(exp2-exp1)/32])     #
#                       si exp2 >= exp1 (sinon echanger r1 et r2)  #
#                                                             #
#=============================================================#

_addrr: link    a6,#-16
        moveml  d2-d7/a2-a4,sp@-
        movl    a6@(8),a2       | a2 pointe sur r2
        movl    a6@(12),a1      | a1 pointe sur r1
        tstb    a2@(4)
        bne     1$
                                | ici r2 = 0 (r2 + r1 = r1)
4$:     tstb    a1@(4)
        bne     22$
                                | ici r2=r1=0
        movl    a1@(4),d1
        cmpl    a2@(4),d1
        bgt     23$
```

```
        movl    a2@(4),d1       | d1.l contient sup(fexp1,fexp2)
23$:    moveq   #3,d0
        bsr     getr
        movl    a0,a6@(-8)
        movl    d1,a0@(4)
        clrl    a0@(8)
        bra     addrrf
                                | ici r2 = 0 et r1 <> 0
22$:    moveq   #0,d0
        movl    a2@(4),d2       | d2.l contient fexp2
        movl    a1@(4),d1
        andl    #0xffffff,d1    | d1.l contient fexp1
        subl    d2,d1           | d1.l recoit exp1-exp2
        bcc     24$
                                | ici exp2 > exp1
        moveq   #3,d0
        bsr     getr
        movl    a0,a6@(-8)      | le resultat est 0 avec exposant fexp2
        movl    a2@(4),a0@(4)
        clrl    a0@(8)
        bra     addrrf
                                | ici exp2 <= exp1
24$:    lsrl    #5,d1           | d1.l contient [(exp1-exp2)/32]
        movw    a1@(2),d0
        subqw   #2,d0           | d0.l contient L1
```

```
        cmpl    d1,d0
        ble     25$
        movl    d1,d0           | d0.l=inf(L1,[(e1-e2)/32])=L
        addql   #1,d0           | le resultat est r1 en longueur:
25$:    addql   #2,d0           | L1 si L1<=[(e1-e2)/32] ou
        bsr     getr
        movl    a0,a6@(-8)
        addql   #4,a1
        addql   #4,a0
        subqw   #2,d0
27$:    movl    a1@+,a0@+
        dbra    d0,27$
        bra     addrrf
                                | ici r2 <> 0
1$:     tstb    a1@(4)
        bne     3$
                                | ici r1 = 0 (r2 + r1 = r2)
        exg     a2,a1
        bra     22$
                                | ici r1 * r2 <> 0
3$:     movb    a1@(4),d3
        movb    a2@(4),d5
        eorb    d5,d3           | d3.b contient : 0 si r1 * r2 > 0
                                | et est negatif sinon
        movb    d3,a6@(-2)      | sauvegarde du 'signe'
```

```
        movl    a2@(4),d3
        andl    #0xffffff,d3    | d3.l contient fexp2=e2
        movl    a1@(4),d1
        andl    #0xffffff,d1    | d1.l contient fexp1=e1
        subl    d1,d3           | d3.l  contient exp2-exp1
        beq     5$              | si e2 = e1
        bcc     6$              | si e2 > e1
                                | ici e2 < e1
        exg     a1,a2
        negl    d3              | d3.l recoit e1-e2 > 0
                                | ici e2-e1 > 0
6$:     movw    d3,d4
        andw    #31,d4
        lsrl    #5,d3           | e2-e1=32*L3+r ; d4.w,d3.l recoit r,L3
        moveq   #0,d2
        movw    a2@(2),d2
        subqw   #2,d2           | d2.l recoit L2
        cmpl    d2,d3
        bcs     7$
                                | ici L3 >= L2 (r1 + r2 = r2)
        movw    a2@(2),d0
        bsr     getr
        movl    a0,a6@(-8)
        addql   #4,a2
        addql   #4,a0
```

```
        subqw   #2,d0
28$:    movl    a2@+,a0@+
        dbra    d0,28$
        bra     addrrf
                                | ici L3 < L2
7$:     moveq   #0,d1
        movw    a1@(2),d1
        subqw   #2,d1           | d1.l recoit L1
        movl    d3,d5
        addl    d1,d5           | d5.l recoit L1 + L3
        cmpl    d2,d5
        bcs     8$              | si L1 + L3 < L2
                                | ici L3 < L2 <= L1 + L3
        movb    #1,a6@(-4)      | a6@(-4) flag contenant :
                                | 0 si L1+L3 < L2 faire alors copie r1
                                | 1 si L3 < L2 <= L1+L3 et idem
                                | 2 si e1 = e2 et alors pas de copie
        movw    d2,d0
        addqw   #2,d0           | d0.w recoit l2
        bsr     getr            | allocation L2+2 lgmots pour resultat
        movl    a0,a6@(-8)      | adresse resultat dans var. locale
        movw    d2,d5
        subw    d3,d5           | d5.w contient L2 - L3
        movw    d5,d0
        addqw   #1,d0           | d0.w contient L2 - L3 + 1
```

```
        bsr     getr            | allocation L2-L3+1 pour copie r1 avec
                                | un unique longmot code
        subqw   #2,d0           | d0.w contient L2 - L3 - 1
        movw    a2@(2),d1
        lea     a2@(0,d1:w:4),a2| a2 pointe fin de r2
        bra     9$
                                | ici L1 + L3 < L2
8$:     clrb    a6@(-4)         | a6@(-4) mis a 0
        movw    d5,d0
        addqw   #3,d0           | d0.w contient L1 + L3 + 3
        bsr     getr            | allocation pour resultat
        movl    a0,a6@(-8)      | adresse resultat dans var. locale
        lea     a2@(0,d0:w:4),a2| a2 pointe ou necessaire !!
        movw    a1@(2),d5       | d5.w contient L1 + 2
        movw    d5,d0           | d0.w contient L1 + 2
        subqw   #2,d5           | d5.w contient L1
        bsr     getr            | allocation L1+2 pour copie r1 avec
                                | un  seul lgmot code
        subqw   #3,d0           | d0.w contient L1 - 1
9$:     movl    a0,a6@(-12)     | adresse copie r1 dans var. locale
        addql   #4,a0
        movl    a0,a3           | a0 et a3 pointent sur debut copie
        addql   #8,a1           | a1 pointe debut mantisse r1
29$:    movl    a1@+,a0@+
        dbra    d0,29$          | boucle copie r1
```

```
        tstw    d4              | test de r = nb de shifts
        bne     10$
                                | ici r = 0 ; pas de shift a faire
                                | a0 pointe fin copie r1
                                | a3 pointe debut mantisse copie r1
        moveq   #0,d7
        movw    a3@(-2),d7
        subqw   #1,d7           | d7.w contient longueur mantisse copie
        movw    d7,d2
        subqw   #1,d2           | d2.w = compteur boucle addition
        lea     a3@(0,d7:w:4),a3| a3 pointe fin copie r1
        movl    a3,a1           | a1 aussi
        bra     11$
                                | ici r <> 0 ; shift a faire
10$:    subqw   #1,d5
        movew   d5,d2           | d5.w et d2.w = compteur boucle shift
        movl    #-1,d6
        lsrl    d4,d6           | masque de shift:0...01...1; avec r '0'
        moveq   #0,d0
                                | boucle de shift de copie de r1
12$:    movl    a3@,d7
        rorl    d4,d7
        movl    d7,d1
        andl    d6,d1
        subl    d1,d7
```

```
        addl    d1,d0
        movl    d0,a3@+
        movl    d7,d0
        dbra    d5,12$
        movl    a3,a1
        tstb    a6@(-4)
        bne     11$             | si a6@(-4) <> 0
                                | ici a6@(-4) = 0
        movl    d0,a1@+
        addqw   #1,d2           | d2.w = compteur boucle addition
11$:    movl    a6@(-8),a0      | a0 pointe sur resultat
        moveq   #0,d1
        movw    a0@(2),d1
        lea     a0@(0,d1:w:4),a0| a0 pointe fin du resultat
        bra     14$
                                | ici e1 = e2
5$:     movb    #2,a6@(-4)      | a6@(-4) recoit 2
        movl    d1,a6@(-16)     | a6@(-16) recoit e1=e2 biaise
        movw    a1@(2),d0
        cmpw    a2@(2),d0
        bcs     15$
        movw    a2@(2),d0
15$:    bsr     getr            | allocation inf (l1,l2) pour resultat
        movl    a0,a6@(-8)      | adresse du resultat dans var. locale
        moveq   #0,d2
```

```
        movw    d0,d2
        movl    d2,d0
        subqw   #3,d2
        moveq   #0,d3
        movl    a2,a4
        movl    a1,a3
        lea     a0@(0,d0:w:4),a0| a0 pointe fin resultat
        lea     a1@(0,d0:w:4),a1| a1 pointe fin de r1 ou copie
        lea     a2@(0,d0:w:4),a2| a2 pointe fin de r2

                                | zone des boucles d'addition

                                | conditions initiales :
                                | a0 pointe fin resultat
                                | a1 pointe fin r1 ou copie
                                | a2 pointe fin r2
                                | d2.w contient L4-1
                                | d3.w contient L3 avec L3+L4=long.res.
14$:    subl    d4,d4           | initialisation bit X
        tstb    a6@(-2)         | test du signe de r1*r2
        bne     surr
                                | ici r1 * r2 > 0
                                | 1ere boucle d'addition
16$:    movl    a1@-,d1
        movl    a2@-,d5
```

```
        addxl   d5,d1
        movl    d1,a0@-
        dbra    d2,16$
        roxrw   d4,d0           | remise a jour du bit C
        bcc     17$             | si pas de carry
        bra     18$             | si carry
                                | 2eme boucle:propagation carry
19$:    movl    a2@-,d5
        addxl   d4,d5
        movl    d5,a0@-
        roxrw   d4,d0           | mise a jour bit C
18$:    dbcc    d3,19$
        bcs     20$             | si carry finale
        bra     17$
                                | 3eme boucle:recopie reste mantisse r2
30$:    movl    a2@-,a0@-
17$:    dbra    d3,30$
        movl    a2@-,a0@-       | mise signe et exposant:celui de r2
        cmpb    #2,a6@(-4)
        beq     addrrf          | si a6@(-4) = 2
                                | ici rendre copie de r1
        movl    a6@(-12),a0
        bsr     giv
        bra     addrrf
                                | ici carry finale
```

```
20$:    movl    a2@-,d1
        andl    #0xffffff,d1
        addql   #1,d1           | d1.l recoit fexp resultat
        cmpl    #0x1000000,d1
        blt     2$
                                | ici fexp>=2^24 : erreur
        movl    #adder4,sp@-
        jsr     _pari_err
                                | ici non debordement
2$:     cmpb    #2,a6@(-4)
        beq     13$
                                | ici rendre copie de r1
        movl    a0,a3
        movl    a6@(-12),a0
        bsr     giv
        movl    a3,a0
13$:    movl    d1,a0@(-4)
        movb    a2@,a0@(-4)     | mise a jour exp et sign resultat
        movw    a0@(-6),d2
        subqw   #3,d2           | compteur de shift
        movw    #-1,d0
        movw    d0,cc           | mise a 1 des bit x et c
31$:    roxrw   a0@+
        roxrw   a0@+            | boucle de mise de retenue finale et
        dbra    d2,31$          | shift de 1 vers la droite mantisse
```

```
addrrf: movl    a6@(-8),d0      | d0 pointe sur resultat
        moveml  sp@+,d2-d7/a2-a4
        unlk    a6
        rts
                                | ici faire une soustraction
                                | pour conditions initiales cf.plus haut
surr:   moveq   #0,d6
        movw    d2,d6
        movw    d2,d7
        addw    d3,d7
        addqw   #3,d7
        cmpb    #2,a6@(-4)
        bne     1$
                                | ici e2 = e1:comparer les mantisses
        addql   #8,a3
        addql   #8,a4
12$:    cmpml   a3@+,a4@+
        dbne    d2,12$
        bhi     1$              | si |r2| > |r1|
        bne     2$              | si |r2| < |r1|
                                | ici |r2| = |r1| et donc r2 + r1 = 0
        movl    a6@(-8),a0      | le resultat est 0 avec comme exposant
        moveq   #0,d2           | -32*inf(l1,l2)+e1
        movw    a0@(2),d2
        subqw   #2,d2
```

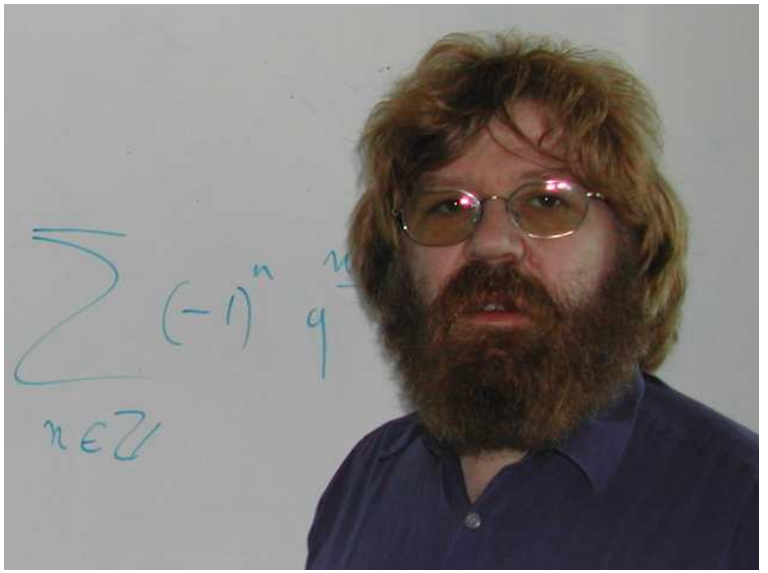
```
        lsll    #5,d2
        negl    d2
        addl    a6@(-16),d2     | ajouter e1 biaise
        bpl     15$
        movl    #adder5,sp@-    | underflow dans R+R
        jsr     _pari_err
15$:    cmpl    #0x1000000,d2
        blt     16$
                                | ici fexp>=2^24 : erreur overflow dans R+R
        movl    #adder4,sp@-
        jsr     _pari_err
16$:    bsr     giv
        moveq   #3,d0
        bsr     getr
        movl    a0,a6@(-8)
        movl    d2,a0@(4)
        clrl    a0@(8)
        bra     addrrf
                                | ici |r2| < |r1| : echanger r2 et r1
2$:     exg     a1,a2
                                | ici |r2| > |r1|
1$:     subw    d2,d6
        subl    d4,d4           | initialisation bit X
                                | 1ere boucle de soustraction
3$:     movl    a2@-,d0
```

```
        movl    a1@-,d5
        subxl   d5,d0
        movl    d0,a0@-
        dbra    d2,3$
        roxrw   d4,d0           | remise ajour bit C
        bra     4$
                                | 2eme boucle:propagation carry
5$:     movl    a2@-,d5
        subxl   d4,d5
        movl    d5,a0@-
        roxrw   d4,d0
4$:     dbcc    d3,5$
        bra     6$
                                | 3eme boucle:copie reste mantisse r2
13$:    movl    a2@-,a0@-
6$:     dbra    d3,13$
        moveq   #0,d3
        moveq   #-1,d2
        movw    d2,d3
14$:    tstl    a0@+
        dbne    d2,14$          | chasse aux '0' du resultat provisoire
                                | a0 pointe sur 1er lgmot non nul
        subw    d2,d3           | d3.w  contient de lgmots nuls
        addw    d6,d3
        subl    #12,a0          | a0 pointe sur resultat
```

```
        movl    a0,a6@(-8)
        movl    a0,a1           | a1 aussi
        cmpb    #2,a6@(-4)
        beq     7$              | si pas de copie faite
                                | ici rendre copie
        movl    a6@(-12),a0
        bsr     giv
7$:     moveq   #0,d0
        movw    d3,d0
        lsll    #2,d0           | d0.l = nb d'octets a 0 du result.
        addl    d0,_avma        | mise a jour pile PARI(rendre d3 lgmot)
        movl    a1,a0           | a0 pointe sur resultat final
        movw    #0x200,a0@
        subw    d3,d7
        movw    d7,a0@(2)       | mise a jour 1er lgmot code resultat
        lsll    #5,d3
        movl    a0@(8),d0
        bfffo   d0{#0:#0},d1    | d1.l contient nb de shifts=r
        lsll    d1,d0           | normalisation 1er lgmot mantisse
        addl    d1,d3
        lsll    #2,d6
        subl    d6,a2
        movl    a2@(-4),d2
        andl    #0xffffff,d2
        subl    d3,d2
```

```
        movl    d2,a0@(4)       | calcul et mise exposant resultat
        movb    a2@(-4),a0@(4)  | mise signe resultat
        tstb    d1
        bne     8$              | si r <> 0
        bra     9$              | si r = 0
8$:     moveq   #1,d6
        lsll    d1,d6
        subql   #1,d6           | masque de shift
        addql   #8,a1
        subqw   #3,d7           | d7.w  contient L-1
        bra     10$
                                | boucle de shift vers la gauche
11$:    movl  a1@(4),d2
        roll    d1,d2
        movl    d2,d3
        andl    d6,d3
        subl    d3,d2
        addl    d3,d0
        movl    d0,a1@+
        movl    d2,d0
10$:    dbra    d7,11$
        movl    d0,a1@
9$:     bra     addrrf
```

The interpreter



GP Parser written by Dominique Bernardi

```
#typedef GEN *long

sizeof(long) == sizeof(void*)
```

Design of a small calculator **GC**

- to call library functions
- to test all modules

First users: Bergé, Martinet and Olivier (sextic fields), Cougnard and Fleckinger.

Batut's PhD thesis

ASPECTS ALGORITHMIQUES DU SYSTÈME

DE CALCUL ARITHMÉTIQUE EN MULTIPRÉCISION

PARI

1er Février 1989

J. Martinet	Président
H. Cohen	Examinateur
F. Dress	Examinateur
Ph. Flajolet	Examinateur

‡

Batut's PhD thesis

Appendix 2: GC User's manual (by H. Cohen)

In its final version, it will be possible to use PARI in three different ways:

Batut's PhD thesis

Appendix 2: GC User's manual (by H. Cohen)

In its final version, it will be possible to use PARI in three different ways:

1) as a library;

Batut's PhD thesis

Appendix 2: GC User's manual (by H. Cohen)

In its final version, it will be possible to use PARI in three different ways:

1) as a library;

2) as a specific language, extension to an upper-level language (C, C++, Pascal or Fortran).

Batut's PhD thesis

Appendix 2: GC User's manual (by H. Cohen)

In its final version, it will be possible to use PARI in three different ways:

1) as a library;

2) as a specific language, extension to an upper-level language (C, C++, Pascal or Fortran).

3) as a sophisticated calculator, named GC.

Batut's PhD thesis

Appendix 2: GC User's manual (by H. Cohen)

In its final version, it will be possible to use PARI in three different ways:

1) as a library;

2) as a specific language, extension to an upper-level language (C, C++, Pascal or Fortran).

3) as a sophisticated calculator, named GC.

At present, only 1) and 3) have been implemented. [. . .]

Appendix 2

its kernel . . . is written in MC68020 assembly language. The rest is at present entirely written in C (although we are planning to use C++ later)

Appendix 2

its kernel . . . is written in MC68020 assembly language. The rest is at present entirely written in C (although we are planning to use C++ later)

1.3.5. Arithmetic functions. *. . . However in the present version 1.11, no efficient factorization method has been implemented, hence the size of the second largest prime divisor of the numbers is for the moment limited to 100000*

Appendix 2

its kernel . . . is written in MC68020 assembly language. The rest is at present entirely written in C (although we are planning to use C++ later)

1.3.5. Arithmetic functions. *. . . However in the present version 1.11, no efficient factorization method has been implemented, hence the size of the second largest prime divisor of the numbers is for the moment limited to 100000*

The present manual is written for version 1.11, but only details will change (like new and faster functions, less bugs, better output, etc. . .) until the next major revision 2.00.

I or **i**: the complex number $\sqrt{-1}$.

K or **k**: level 0 formal parameter for sums, products and recursions.

L or **l**: level 1 formal parameter for sums, products and recursions.

U or **u**: the universal sequence used in recursions

W or **w**: $\sqrt{d}/2$ or $(1+\sqrt{d})/2$

X, Y, Z, T, A, B, C, D or **x, . . . , d**: the only permitted names of the formal variables of a polynomial, power series, polymod, or rational functions.

3.5.11. roots(x): ... The algorithm used is a variant of the Newton-Raphson method and is not guaranteed to converge. If you get the message “too many iterations in roots” or “INTERNAL ERROR: incorrect result in roots”, try modifying your polynomial to get the roots.

gerepile

Apart from universal objects (see below) the chunks of memory used by a given PARI object must be in consecutive memory locations

gerepile

Apart from universal objects (see below) the chunks of memory used by a given PARI object must be in consecutive memory locations

```
GEN gerepile(long ltop, long lbot, GEN z);
```

The understanding of the behavior of this function is certainly the most difficult part of this chapter, hence we will try to go slowly in the explanation.

...

If you followed us this far, congratulations, and rejoice, the rest is must easier. [...] Even for the authors, the use of gerepile was not evident at first.

First release

PARI 1.31 released on February 28, 1990 (sci.math)

Available by ftp from `mizar.greco-prog.fr`

> On a SUN 3/xxx, it is between 5 to 100 times faster than Maple or Mathematica [...]
>
> To get the MacII or Mac SE/30 version, send an email message to bernardi@mizar.greco-prog.fr [...]
>
> Please understand that it is not a polished commercial product.

At the same time. . .

1st March 1990: From Pierrette Cassou (sci.math)

The mathematical research potential of Universite Bordeaux I is in danger. The number of students is exponentially increasing, yet the number of faculty and staff (as fixed by the beaureacrats in Paris) is stable. We find ourselves obliged to increase our teaching loads each semester to the detriment of our research and our students.

Unable to staff several courses, we have declared a general suspension of instruction in mathematics for the last 22 days. This Tuesday, several hundred students and faculty will assemble in Paris in order to ask the minister of education for new positions.

We ask for your support by sending a telegram or fax to:
Ministere de l'Education Nationale
110, rue de Grenelle
75007 Paris France
Fax: (33) (1) 45 55 15 56

Problems with Mizar

The `mizar.greco-prog.fr` machine is not reliable.

A mirror is setup at `seti.inria.fr`:

`sci.math, Mar 17 1990, 9:45 pm`

```
I decided to post, since other people might have the same question.
Pari is accessible from another site.
Try ftp at 128.93.1.12 (seti.inria.fr)
I've been looking through the code a bit.  It's great stuff,
and it's got everything.
I have been trying to port it to my PC, which has been challenging,
but I am making good progress.
It compiled and executed today, but I have no idea if it works, since
I haven't yet read the manual, and I don't know what to do.
I've been waiting a long time for public domain routines like these.
Thanks much to the authors.

        Karl Dahlke     ekl...@brahms.berkeley.edu
```

Success again . . .

Richard Fateman (1991):

> *For Pari 1.35.01 the time for a comparable multiplication* $[\text{squaring a } 735\text{-digit value of } \pi]$ *and storage of the result was about 15 ms.* $[\text{for Pari } 1.35.01]$*, for Mathematica 2.2 about 29 ms., for Maple V about 83.5 ms.*

. . . and again

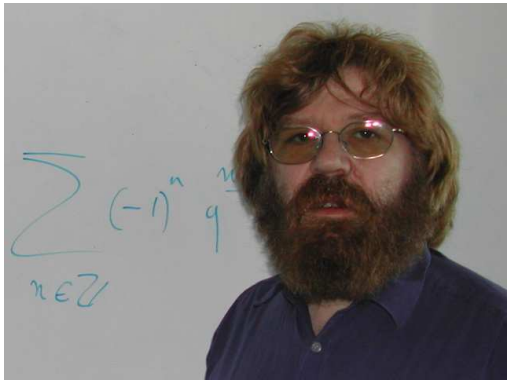
MSc thesis of Thomas W. Mattman, *The computation of Galois groups over function fields*, McGill University, December 1992:

> *We compared the MAPLE V [CGG] and PARI 1.36 [BBCO] implementations of LLL on a SUN 3/50 for the case of* $\mathrm{SL}(2,3)^+$. *In PARI, LLL takes 10 seconds of CPU if real arithmetic is used, and 1500 seconds for rational arithmetic. Although the two versions came up with the same result in this case, in general, the real arithmetic version is 'numerically unstable' (see [BBCO, p.30]). The MAPLE implementation requires 2040 seconds of CPU and uses rational arithmetic.*

Pari 1.36

Pari-1.36, released in December 1991.

Copyright BaBe CoOl


Porting on 64-bit machines

From: LBARTHO@uni2a.unige.ch

Je viens de me pencher sur le probleme de la construction interne de PARI. Cela pose d'enormes problemes pour adapter le code a alpha, car alpha est un processeur 64 bits et PARI suppose partout que les nombres et les pointeurs tiennent sur 32 bits.

Certainement qu'un ensemble d'etudiants et d'assistants pourraient rendre le code plus portable? quelques conseils:

- le type d'un objet est une *enumeration*
- un objet est une *structure*
- les endroits ou une taille est fixee (ex. 32 bits) sont signales ainsi, les extensions sont plus faciles a faire.
- Ne pas reinventer la roue: la librairie gmp (Gnu Multi Precision) est disponible, gratuite, tres rapide, et bien testee
- le C++ permet des constructions plus elegantes: des objets peuvent etre automatiquement alloues sur la pile, la pile peut etre nettoyee automatiquement en fin d'appel, et les operatuers de base peuvent etre utilises en mode infixe.

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

7 DESS (Génie Logiciel) students, very professional approach

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

7 DESS (Génie Logiciel) students, very professional approach

Henri provides Sun4 machines, and a Sony laptop

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

7 DESS (Génie Logiciel) students, very professional approach

Henri provides Sun4 machines, and a Sony laptop

implementation in C++, user-language near Pascal

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

7 DESS (Génie Logiciel) students, very professional approach

Henri provides Sun4 machines, and a Sony laptop

implementation in C++, user-language near Pascal

> *We had great difficulties distinguishing PARI functions from GP functions. That was even worse when it came out that the PARI sources mentioned some GP data structures. [. . .] The use of non-significant variable names did not ease our work.*

The MOLI project (92-93)

MOLI = Mathematic Object Language Interpreter

7 DESS (Génie Logiciel) students, very professional approach

Henri provides Sun4 machines, and a Sony laptop

implementation in C++, user-language near Pascal

> *We had great difficulties distinguishing PARI functions from GP functions. That was even worse when it came out that the PARI sources mentioned some GP data structures. [. . .] The use of non-significant variable names did not ease our work.*

```
GEN classno2(x) GEN x;
{
  long av=avma,tetpil,n,i,k,s=signe(x),fl2,ex;
  GEN p1,p2,p3,p4,p5,p6,p7,p9,pi4,d,reg,logd;
```

```
pari-1.36$ grep RAVYZARC *.h
extern  GEN     RAVYZARC;
#define isonstack(x)   (RAVYZARC=(GEN)(x),((RAVYZARC>=(GEN)bot)&&\
              (RAVYZARC<(GEN)top)))
#define copyifstack(x) (RAVYZARC=(GEN)(x),((RAVYZARC>=(GEN)bot)&&\
              (RAVYZARC<(GEN)top))?lcopy(RAVYZARC):(long)RAVYZARC)
#define adecaler(x,tetpil,anavma) (RAVYZARC=(GEN)(x),((RAVYZARC>=\
              (GEN)anavma)&&(RAVYZARC<(GEN)tetpil)))
```

```
pari-1.36$ grep RAVYZARC *.h
extern  GEN     RAVYZARC;
#define isonstack(x)   (RAVYZARC=(GEN)(x),((RAVYZARC>=(GEN)bot)&&\
              (RAVYZARC<(GEN)top)))
#define copyifstack(x) (RAVYZARC=(GEN)(x),((RAVYZARC>=(GEN)bot)&&\
              (RAVYZARC<(GEN)top))?lcopy(RAVYZARC):(long)RAVYZARC)
#define adecaler(x,tetpil,anavma) (RAVYZARC=(GEN)(x),((RAVYZARC>=\
              (GEN)anavma)&&(RAVYZARC<(GEN)tetpil)))

pari-1.36$ grep yatileugoto *.c
anal.c:  static long yatileugoto;
anal.c:   analyseurs=labellist[m];yatileugoto=1;break;
anal.c:       yatileugoto=0;res = seq();
anal.c:       if(!yatileugoto) {match(',');skipseq();}
anal.c:       yatileugoto=0;seq();
anal.c:       if(!yatileugoto) analyseurs = ch1;else break;
```

MOLI Features

1. block structures,
2. routines without side-effect,
3. functions and procedures differ,
4. distinguish local and global variables,
5. distinguish upper/lower case,
6. uniformization of builtin and user functions,
7. distinguish expression and result variables.

GP/PARI everywhere

March 1992: VAX/VMS port (David Ford and Michel Olivier)

Sep 1993: Amiga port (Niels Moller)

Nov 1993: `parimail@ceremab.u-bordeaux.fr` (128 subscribers)

1994: Risa/Asir uses PARI (Noro and Takeshima, Fujitsu Labs)

1994: MuPAD uses PARI (Benno Fuchssteiner, Univ. Paderborn)

Also used by Magma, GCL (GNU Common Lisp), Maxima (Bill Schelter).

BUG ALERT

Date: Tue, 1 Mar 1994 13:47:48 -0500
From: Henri Cohen <cohen@ceremab.u-bordeaux.fr>
Subject: BUG ALERT IN PARI/GP and AXIOM
Newsgroup: sci.math.research

To all PARI/GP users (AXIOM users read at the end)

A nasty bug in the division routine used in the Pari Kernel has been found by Bill Dubuque. We inform you immediately so that you can take appropriate measures concerning your own software which may use the Pari kernel. A new release of Pari will of course be forthcoming very quickly to repair this bug, but in the meantime you should correct it. The bug can be detected as follows:

```
b=2^32;n=(b-1)*(b^2+1);d=b^2-1;
(n*1./d)*d-n
```

The final result should be close to 0. It is not.

Important Dates

1995: K. Belabas takes over the development of PARI

Important Dates

1995: K. Belabas takes over the development of PARI

Nov 1997: two mailing lists : users, developers (300 subscribers)

Important Dates

1995: K. Belabas takes over the development of PARI

Nov 1997: two mailing lists : users, developers (300 subscribers)

Feb 1998: DOS/OS2/Windows port of pari-2.0.x.alpha (Bruno Haible) and PowerMac port (D. Bernardi)

Important Dates

1995: K. Belabas takes over the development of PARI

Nov 1997: two mailing lists : users, developers (300 subscribers)

Feb 1998: DOS/OS2/Windows port of pari-2.0.x.alpha (Bruno Haible) and PowerMac port (D. Bernardi)

Sep 1999: public CVS server, stable and unstable branches

Important Dates

1995: K. Belabas takes over the development of PARI

Nov 1997: two mailing lists : users, developers (300 subscribers)

Feb 1998: DOS/OS2/Windows port of pari-2.0.x.alpha (Bruno Haible) and PowerMac port (D. Bernardi)

Sep 1999: public CVS server, stable and unstable branches

July 2000: RMS comes at RMLL'2000

Important Dates

1995: K. Belabas takes over the development of PARI

Nov 1997: two mailing lists : users, developers (300 subscribers)

Feb 1998: DOS/OS2/Windows port of pari-2.0.x.alpha (Bruno Haible) and PowerMac port (D. Bernardi)

Sep 1999: public CVS server, stable and unstable branches

July 2000: RMS comes at RMLL'2000

Nov 2000: PARI becomes GPL (version 2.1.0)

Down Side

Pari 1.xx was ill-designed, with a user-unfriendly interface

Down Side

Pari 1.xx was ill-designed, with a user-unfriendly interface

Pari 2.xx: cleanup the implementation, keep the (bad) user language

Down Side

Pari 1.xx was ill-designed, with a user-unfriendly interface

Pari 2.xx: cleanup the implementation, keep the (bad) user language

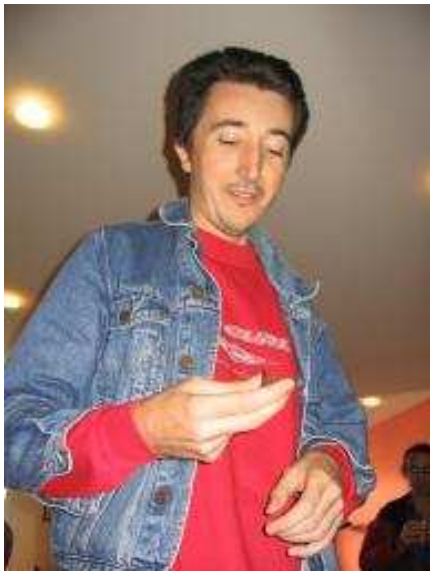
Pari 3.xx: cleanup the user language?

Down Side

Pari 1.xx was ill-designed, with a user-unfriendly interface

Pari 2.xx: cleanup the implementation, keep the (bad) user language

Pari 3.xx: cleanup the user language?



Up Side

- despite a bad user-language (GP vs MOLI);

Up Side

- despite a bad user-language (GP vs MOLI);
- despite a crazy implementation (C vs C++);

Up Side

- despite a bad user-language (GP vs MOLI);
- despite a crazy implementation (C vs C++);
- despite a dummy business model (GPL vs commercial);

Up Side

- despite a bad user-language (GP vs MOLI);
- despite a crazy implementation (C vs C++);
- despite a dummy business model (GPL vs commercial);
- despite (almost) no GUI;

Up Side

- despite a bad user-language (GP vs MOLI);
- despite a crazy implementation (C vs C++);
- despite a dummy business model (GPL vs commercial);
- despite (almost) no GUI;

GP/PARI is still alive!

THANK YOU HENRI

FOR GP/PARI