```
%
%       fap_n.sm4
%
%       MMIX assembly code for low level routines of
%       fast arbitrary precision arithmetic of natural numbers.
%
%       Arbitrary precision natural numbers are represented as a
%       sequence of words.
%
%       A word is an octabyte on MMIX. The most significant word is
%       on the highest address, and it is always supposed to be non-zero.
%
%       Running times here are the running time of the routine,
%       without the setting up of parameters by callee routine
%       and the time of the POP instruction for return.
%
%       Code length is the number of tetras for the routine, not
%       counting (one) POP instruction for return.
%
%
%



%       THE CLASSICAL ALGORITHMS



%
        PREFIX  :leading_zeros:
%
%       (third-rank candidate to rewrite in assembly)
%
% This simple routine calculate the number of leading zeros of
% the nonzero input word.
%
%
%       int leading_zeros(w)
%
%       Word    w;      //input word, <>0
%
%
% Entry:
%
w       IS      $0      %       the (nonzero) input word
%
% Exit: the number of leading zeros, >=0 and <word_size
%
%
% Profile:
%
% w=    #ffffffffffffffff                #7fffffffffffffff
%       4,0,7;0,0                        4,0,7;0,0
%
%
% w=    #3fffffffffffffff                #07ffffffffffffff
%       4,0,7;0,0                        4,0,7;0,0
%
%
% w=    #003fffffffffffff                #00000fffffffffff
%       4,0,7;0,0                        4,0,7;0,0
%
%
% w=    #0000000000003fff
%       4,0,7;0,0
%
%



%
        PREFIX  :trealing_zeros:
%
```



```
%       (third-rank candidate to rewrite in assembly)
%
% This simple routine calculate the number of trealing zeros of
% the nonzero input word.
%
%
%       int trailing_zeros(w)
%
%       Word    w;      //input word, <>0
%
%
% Entry:
%
w       IS      $0      %       the (nonzero) input word
%
% Exit: the number of trealing zeros, >=0 and <word_size
%
%
% Profile:
%
%       2 instructions
%
% w=    #ffffffffffffffff                #fffffffffffffffe
%       2,0,2;0,0                        2,0,2;0,0
%
% w=    #fffffffffffffffc                #ffffffffffffffe0
%       2,0,2;0,0                        2,0,2;0,0
%
% w=    #fffffffffffffc00                #ffffffffff00000
%       2,0,2;0,0                        2,0,2;0,0
%
% w=    #fffc000000000000
%       4,0,7;0,0
%
%

%
        PREFIX  :ap_l_copy:
%
%       (third-rank candidate to rewrite in assembly)
%
%       This simple routine copies an arbitrary precision number.
%
%
%       void ap_l_copy(d,s,l)
%
%       Word    *d;     // pointer to the LSW of the destination
%       Word    *s;     // pointer to the LSW of the source
%       Size    l;      // length of source in words; always nonnegative
%
%
% Entry:
%
%
d       IS      $0      % pointer to the LSW of the destination
s       IS      $1      % pointer to the LSW of the source
l       IS      $2      % length of source in words; always nonnegative.
%
%
% Exit: -
%
%
% Profile:
%
%       44 instructions
%
% l=    0                                1
%       10,0,14;0,1                      12,2,16;0,1
%
% l=    2                                5
```

```
%       14,4,18;0,1                         20,10,24;0,1
%
% l=    10                                  20
%       30,20,34;0,1                        54,40,58;1,1
%
% l=    50                                  100
%       122,100,126;3,1                     224,200,238;6,1
%
% l=    200                                 500
%       458,400,462;12,1                    1134,1000,1138;31,1
%


%
        PREFIX  :ap_l_lsh:
%
%       (third-rank candidate to rewrite in assembly)
%
%       This simple routine performs a left shift of an arbitrary
%       precision natural number.  The destination and the source pointer
%       may be the same.
%
%
%       Size ap_l_lsh(d,s,l,u)
%
%       Word    *d;     // pointer to the LSW of the destination
%       Word    *s;     // pointer to the LSW of the source
%       Size    l;      // length of source in words; always nonnegative
%       Word    u;      // number of bits to shift; always nonnegative
%
%
% Entry:
%
d       IS      $0      % pointer to the LSW of the destination
s       IS      $1      % pointer to the LSW of the source
l       IS      $2      % length of source in words; always nonnegative
u       IS      $3      % number of bits to shift; always nonnegative
%
% Exit: The length of the result in words
%
%
% Profile:
%
%       26 instructions
%
%       l=0,    u=1
%       2,0,4;0,1
%
%       l=1,    u=1
%       21,2,27;1,3
%
%       l=2,    u=1
%       28,4,34;2,3
%
%       l=5,    u=1
%       49,10,55;5,3
%
%       l=10,   u=1
%       84,20,90;10,3
%
%       l=20,   u=1
%       154,40,160;20,3
%
%       l=50,   u=1
%       364,100,370;50,3
%
%       l=100,  u=1
%       714,200,720;100,3
%
%       l=200,  u=1
```



```
%       1414,400,1420;200,3
%
%       l=500,  u=1
%       3514,1000,3520;500,3
%
%       l=1,    u=100
%       27,3,33;2,3
%
%       l=1,    u=200
%       33,5,39;4,3
%
%       l=1,    u=500
%       45,9,51;8,3
%
%       l=1,    u=1000
%       69,17,75;16,3
%
%       l=1,    u=2000
%       117,33,123;32,3
%
%       l=1,    u=5000
%       258,80,264;79,3
%
%       l=1,    u=10000
%       492,158,498;157,3
%
%       l=1,    u=20000
%       960,314,966;313,3
%
%       l=1,    u=50000
%       2367,783,2373;782,3
%


%
        PREFIX  :ap_l_rsh:
%
%       (third-rank candidate to rewrite in assembly)
%
%       This simple routine performs a right shift of an arbitrary
%       precision natural number.  The destination and the source pointer
%       may be the same.
%
%
%       Size ap_l_rsh(d,s,l,u)
%
%       Word    *d;     // pointer to the LSW of the destination
%       Word    *s;     // pointer to the LSW of the source
%       Size    l;      // length of source in words; always nonnegative
%       Word    u;      // number of bits to shift; always nonnegative
%
%
% Entry:
%
d       IS      $0      %       pointer to the LSW of the destination
s       IS      $1      %       pointer to the LSW of the source
l       IS      $2      %       length of source in words; always nonnegative.
u       IS      $3      %       number of bits to shift; always nonnegative.
%
% Exit: The length of the result in words
%
%
% Profile:
%
%       26 instructions
%
%       l=0,    u=1
%       7,0,9;0,1
%
%       l=1,    u=1
```

```
%       16,1,20;1,2
%
%       l=2,    u=1
%       27,4,29;3,1
%
%       l=5,    u=1
%       48,10,50;6,1
%
%       l=10,   u=1
%       83,20,85;11,1
%
%       l=20,   u=1
%       153,40,155;21,1
%
%       l=50,   u=1
%       363,100,365;51,1
%
%       l=100,  u=1
%       713,200,715;101,1
%
%       l=200,  u=1
%       1413,400,1415;201,1
%
%       l=500,  u=1
%       3513,1000,3515;501,1
%


%
%       ap_l_addc and ap_l_subc routines
%
%       (first-rank candidates to rewrite in assembly)
%
%       These simple routines performs addition and substraction of
%       arbitrary precisions natural numbers having the same length.
%       A carry word (not just one bit!) also added.
%       The destination and the source pointers may be the same
%
%
%       Word    ap_l_addc(c,d,s1,s2,l)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the destination
%       Word    *s1     // pointer to the LSW of the first source
%       Word    *s2     // pointer to the LSW of the second source
%       Size    l       // length of sources in the words, >=0
%
%
%       Word    ap_l_subc(c,d,s1,s2,l)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the destination
%       Word    *s1     // pointer to the LSW of the first source
%       Word    *s2     // pointer to the LSW of the second source
%       Size    l       // length of sources in the words, >=0
%
%
% Entry:
%
c       IS      $0      % incoming carry
d       IS      $1      % pointer to the LSW of the destination
s1      IS      $2      % pointer to the LSW of the first source
s2      IS      $3      % pointer to the LSW of the second source
l       IS      $4      % length of sources in the words, always nonnegative
%
% Exit: the outgoing carry
%
%
% Profile:
%
```



```
%       ap_l_addc
%
%       c=1
%
%       l=0
%       1,0,3;0,1
%
%       l=1
%       27,3,31;2,1
%
%       l=2
%       35,6,39;2,1
%
%       l=5
%       59,15,63;2,1
%
%       l=10
%       99,30,103;2,1
%
%       l=20
%       184,60,188;3,1
%
%       l=50
%       434,150,438;5,1
%
%       l=100
%       849,300,853;8,1
%
%       l=200
%       1679,600,1683;14,1
%
%       l=500
%       4174,1500,4178;33,1
%
%
%       ap_l_subc
%
%       c=1
%
%       l=0
%       1,0,3;0,1
%
%       l=1
%       27,3,31;2,1
%
%       l=2
%       35,6,39;2,1
%
%       l=5
%       59,15,63;2,1
%
%       l=10
%       99,30,103;2,1
%
%       l=20
%       184,60,188;3,1
%
%       l=50
%       434,150,438;5,1
%
%       l=100
%       849,300,853;8,1
%
%       l=200
%       1679,600,1683;14,1
%
%       l=500
%       4174,1500,4178;33,1
%
```

```
%
%       Multiplication and squaring
%

%
        PREFIX  :ap_l_mul11c:
%
%       (first-rank candidate to rewrite in assembly)
%
%       a*m+c is calculated, where a,m and c are words. The LSW of the
%       results is put to the place pointed by d. The MSW is the return value.
%
%
%       Word    ap_l_mul11c(c,d,a,m)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the destination
%       Word    a       // multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % incoming carry
d       IS      $1      % pointer to the LSW of the destination
a       IS      $2      % multiplicand
m       IS      $3      % multiplier
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       7 instructions
%
%       c=4     a=2     m=3
%       7,1,16;0,0
%

%
        PREFIX  :ap_l_mul21c:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (s[0]+s[1]*B)*m+c is calculated, where B=2^64.
%
%
%       Word    ap_l_mul21c(c,d,s,m)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the destination
%       Word    *s      // pointer to the LSW of the multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % incoming carry
d       IS      $1      % pointer to the LSW of the destination
s       IS      $2      % pointer to the LSW of the multiplicand
m       IS      $3      % multiplier
%
%
% Exit: MSW of the result
%
%
% Profile:
```



```
%
%       16 instructions
%
%       c=4     s[0]=1  s[1]=2  c=3
%       16,4,34;0,0
%

%
        PREFIX  :ap_l_mul22cc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (s[0]+s[1]*B)*(m0+m1*B)+c[0]+c[1]*B is calculated, where B=2^64.
%
%
%       Word    ap_l_mul22cc(c,d,s,m0,m1)
%
%       Word    *c      // pointer to the LSW of the two-words incoming carry
%       Word    *d      // pointer to the LSW of the destination
%       Word    *s      // pointer to the LSW of the multiplicand
%       Word    m0      // multiplier LSW
%       Word    m1      // multiplier MSW
%
%
% Entry:
%
c       IS      $0      % pointer to the LSW of the two-words carry
d       IS      $1      % pointer to the LSW of the destination
s       IS      $2      % pointer to multiplicand
m0      IS      $3      % multiplier LSW
m1      IS      $4      % multiplier MSW
%
%
% Exit: c[0] and c[1] is set
%
%
% Profile:
%
%       40 instructions
%
%       c[0]=1001 c[1]=1002 s[0]=1 s[1]=2 m0=3 m1=4
%       40,8,76;0,0
%


%
        PREFIX  :ap_l_mul11addc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       a*b+c+d[0] is calculated, where a,b,c and d[0] are words.
%       The LSW of the result replace d[0] and the MSW of the result
%       is the return value.
%
%
%       Word    ap_l_mul11addc(c,d,a,m)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the source/destination
%       Word    a       // multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % third operand
d       IS      $1      % pointer to the LSW of the destination/source
a       IS      $2      % first operand
b       IS      $3      % second operand
```

```
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       c=4     [d]=1   a=2     b=3
%       12,2,21;0,0
%



%
        PREFIX  :ap_l_mul21addc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (s[0]+s[1]*B)*m+c+d[0]+d[1]*B is calculated, where B=2^64
%       and s[0], s[1], m, c, d[0] and d[1] are words.
%       The result replace d[0], d[1] and the MSW of the result is
%       the return value.
%
%
%       Word    ap_l_mul21addc(c,d,s,m)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of the source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % carry
d       IS      $1      % pointer to the LSW of the destination/source
s       IS      $2      % pointer to the LSW of multiplicand
m       IS      $3      % multiplier
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       26 instructions
%
%       c=4 d[0]=1 d[1]=2 s[0]=1001 s[0]=1002 m=3
%       26,6,44;0,0
%



%
        PREFIX  :ap_l_mul22addcc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (s[0]+s[1]*B)*(m0+m1*B)+c[0]+d[0]+(c[1]+d[1])*B is calculated,
%       where B=2^64 and s[0], s[1], m0, m1, c[0], c[1], d[0] and d[1]
%       are words. The result replace d[0], d[1] and the MSW of the result is
%       the return value.
%
%
%       Void    ap_l_mul22addc(c,d,s,m0,m1)
%
%       Word    *c      // pointer to the LSW of the carries c[0], c[1]
%       Word    *d      // pointer to the LSW of the source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m0      // multiplier, LSW
%       Word    m1      // multiplier, MSW
```



```
%
%
% Entry:
%
c       IS      $0      % pointer to the LSW of the carries c[0], c[1]
d       IS      $1      % pointer to the LSW of the destination/source
s       IS      $2      % pointer to the LSW of multiplicand
m0      IS      $3      % multiplier, LSW
m1      IS      $4      % multiplier, MSW
%
%
% Exit: c[0], c[1] are set
%
%
% Profile:
%
%       48 instructions
%
%       c[0]=2001 c[1]=2002 d[0]=1 d[1]=2 s[0]=1001 s[1]=1002 m0=3 m1=4
%       48,10,84;0,0
%
%



%
        PREFIX  :ap_l_mul1c:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a word m and adds a carry word c. The result
%       is stored at place pointed by d. The length of d and s is l>0.
%       The new carry is returned.
%
%
%       Word    ap_l_mul1(c,d,s,m,l)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m       // multiplier
%       Size    l       // length of source and destination in words
%
%
% Entry:
%
c       IS      $0      % carry word
d       IS      $1      % pointer to LSW of destination
s       IS      $2      % pointer to LSW of source
m       IS      $3      % multiplier
l       IS      $4      % length of source and destination in words
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       14 instructions
%
%       c=0,    m=0
%
%       l=1
%       14,2,25;0,1
%
%       l=2
%       24,4,44;1,1
%
%       l=5
```

```
%       54,10,101;4,1
%
%       l=10
%       104,20,196;9,1
%
%       l=20
%       204,40,386;19,1
%
%       l=50
%       504,100,956;49,1
%
%       l=100
%       1004,200,1906;99,1
%
%       l=200
%       2004,400,3806;199,1
%
%       l=500
%       5004,1000,9506;499,1
%
%       c=0,    m=#ffffffffffffffff
%
%       l=1
%       14,2,25;0,1
%
%       l=2
%       24,4,44;1,1
%
%       l=5
%       54,10,101;4,1
%
%       l=10
%       104,20,196;9,1
%
%       l=20
%       204,40,386;19,1
%
%       l=50
%       504,100,956;49,1
%
%       l=100
%       1004,200,1906;99,1
%
%       l=200
%       2004,400,3806;199,1
%
%       l=500
%       5004,1000,9506;499,1
%
%



%
        PREFIX  :ap_l_mul2:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a two-word number m0+m1*B, where B=2^64.
%       The result is stored at place pointed by d.
%       The length of d and s is l>1.
%       The MSW is returned, the second MSW is stored.
%       The new carry is returned.
%
%
%       Word    ap_l_mul2(d,s,m0,m1,l)
%
%       Word    *d      // pointer to the LSW of destination
%       Word    *s      // pointer to the LSW of multiplicand
```



```
%       Word    m0      // multiplier, LSW
%       Word    m1      // multiplier, MSW
%       Size    l       // length of source and destination in words
%
%
% Entry:
%
d       IS      $0      % pointer to LSW of destination
s       IS      $1      % pointer to LSW of source
m0      IS      $2      % multiplier, LSW
m1      IS      $3      % multiplier, MSW
l       IS      $4      % length of source and destination in words
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       simple version:
%
%       37 instructions
%
%       m0=3    m1=4
%
%       l=2
%       37,5,75;0,1
%
%       l=5
%       94,11,184;3,1
%
%       l=10
%       189,21,371;8,1
%
%       l=20
%       379,41,741;18,1
%
%       l=50
%       949,101,1851;48,1
%
%       l=100
%       1899,201,3701;98,1
%
%       l=200
%       3799,401,7401;198,1
%
%       l=500
%       9499,1001,18501;498,1
%
%
%       m0=3    m1=#ffffffffffffffff
%
%       l=2
%       37,5,75;0,1
%
%       l=5
%       94,11,184;3,1
%
%       l=10
%       189,21,371;8,1
%
%       l=20
%       379,41,741;18,1
%
%       l=50
%       949,101,1851;48,1
%
%       l=100
%       1899,201,3701;98,1
```

```
%
%       l=200
%       3799,401,7401;198,1
%
%       l=500
%       9499,1001,18506;498,1
%



%
        PREFIX  :ap_l_mul1addc:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a word m and adds a carry word c to the
%       product. The result is added to the destination pointed by d.
%       The length of d and s is l, may be 0.
%       The new carry is returned.
%
%
%       Word    ap_l_mul1addc(c,d,s,m,l)
%
%       Word    c       // incoming carry
%       Word    *d      // pointer to the LSW of destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m       // multiplier
%       Size    l       // length of source and destination in words
%
%
% Entry:
%
c       IS      $0      % carry word
d       IS      $1      % pointer to LSW of destination
s       IS      $2      % pointer to LSW of source
m       IS      $3      % multiplier
l       IS      $4      % length of source and destination in words
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       20 instructions
%
%       c=0,    m=0
%
%       l=0
%       1,0,3;0,1
%
%       l=1
%       21,3,32;1,1
%
%       l=2
%       35,6,55;2,1
%
%       l=5
%       77,15,124;5,1
%
%       l=10
%       147,30,239;10,1
%
%       l=20
%       287,60,469;20,1
%
%       l=50
%       707,150,1159;50,1
%
```



```
%       l=100
%       1407,300,2307;100,1
%
%       l=200
%       2807,600,4607;200,1
%
%       l=500
%       7007,1500,11507;500,1
%
%
%       c=0,    m=#ffffffffffffffff
%
%       l=0
%       1,0,3;0,1
%
%       l=1
%       21,3,32;1,1
%
%       l=2
%       35,6,55;2,1
%
%       l=5
%       77,15,124;5,1
%
%       l=10
%       147,30,239;10,1
%
%       l=20
%       287,60,469;20,1
%
%       l=50
%       707,150,1159;50,1
%
%       l=100
%       1407,300,2307;100,1
%
%       l=200
%       2807,600,4607;200,1
%
%       l=500
%       7007,1500,11507;500,1
%
%



%
        PREFIX  :ap_l_mul2add:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a two-word number m0+m1*B, where
%       B=2^64, and adds a carry word c.
%       The result is stored at place pointed by d.
%       The length of d and s is l>1.
%       The MSW is returned, the second MSW is stored.
%
%
%       Word    ap_l_mul2add(d,s,m0,m1,l)
%
%       Word    *d      // pointer to the LSW of destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m0      // multiplier, LSW
%       Word    m1      // multiplier, MSW
%       Size    l       // length of source and destination in words
%
%
% Entry:
%
```

```
d       IS      $0      % pointer to LSW of destination
s       IS      $1      % pointer to LSW of source
m0      IS      $2      % multiplier
m1      IS      $3      % multiplier
l       IS      $4      % length of source and destination in words
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%
%       46 instructions
%
%       m0=3    m1=4    s[0]=1 ...      d[1]=1001 ...
%
%       l=2
%       46,7,84;0,1
%
%       l=5
%       115,16,207;3,1
%
%       l=10
%       230,31,412;8,1
%
%       l=20
%       460,61,822;18,1
%
%       l=50
%       1150,151,2052;48,1
%
%       l=100
%       2300,301,4102;98,1
%
%       l=200
%       4600,601,8202;198,1
%
%       l=500
%       11500,1501,20502;498,1
%
%
%       m0=3    m1=#ffffffffffffffff    s[0]=1 ...      d[1]=1001 ...
%
%       l=2
%       46,7,84;0,1
%
%       l=5
%       115,16,207;3,1
%
%       l=10
%       230,31,412;8,1
%
%       l=20
%       460,61,822;18,1
%
%       l=50
%       1150,151,2052;48,1
%
%       l=100
%       2300,301,4102;98,1
%
%       l=200
%       4600,601,8202;198,1
%
%       l=500
%       11500,1501,20502;498,1
%
```



```
%
        PREFIX  :ap_l_mul_class:
%
%       (third-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s1 with an other arbitrary precision number pointed
%       by s2. The numbers has length l1 and l2, respectively, where
%       l1>=l2>0 is supposed. The least significant l1+l2-1 word of
%       the result is put to the place pointed by d, the MSW is the
%       return value.
%
%
%       Word    ap_l_mul_class(d,s1,s2,l1,l2)
%
%       Word    *d      // pointer to the LSW of destination
%       Word    *s1     // pointer to the LSW of multiplicand
%       Word    *s2     // pointer to the LSW of multiplier
%       Size    l1      // length of multiplicand
%       Size    l2      // length of multiplier
%
%
% Entry:
%
d       IS      $0      % pointer to LSW of destination
s1      IS      $1      % pointer to LSW of first source
s2      IS      $2      % pointer to LSW of second source
l1      IS      $3      % length of first source in words
l2      IS      $4      % length of second source in words
%
% Exit: MSW of the result
%
%
% Profile:
%
%       116 instructions
%
%       l=l1=l2
%
%       l=1
%       26,3,38;1,2
%
%       l=2
%       46,7,88;0,3
%
%       l=3
%       117,20,204;5,3
%
%       l=4
%       179,27,331;5,4
%
%       l=5
%       301,49,534;13,4
%
%       l=6
%       404,59,738;14,5
%
%       l=7
%       577,90,1028;25,5
%
%       l=8
%       721,103,1309;27,6
%
%       l=9
%       945,143,1686;41,6
%
%       l=10
%       1130,159,2044;44,7
```

```
%
%       l=11
%       1405,208,2508;61,7
%
%       l=12
%       1631,227,2943;65,8
%
%       l=13
%       1957,285,3494;85,8
%
%       l=14
%       2224,307,4006;90,9
%
%       l=15
%       2601,374,4644;113,9
%
%       l=16
%       2909,399,5233;119,10
%
%



%
        PREFIX  :ap_l_sqr1:
%
%       (first-rank candidate to rewrite in assembly)
%
%       a*a is calculated, where a is a word. The LSW of the
%       results is put to the place pointed by p, the MSW is the return value.
%
%
%       Word    ap_l_sqr1(d,a)
%
%       Word    *d      // pointer to the LSW of destination
%       Word    a       // input word
%
%
% Entry:
%
d       IS      $0      % pointer to the LSW of the destination
a       IS      $1      % operand
%
% Exit: MSW of the result
%
%
% Profile:
%
%       3 instruction
%
%       a=2     c=3     [d]=1
%       3,1,12;0,0
%
%



%
        PREFIX  :ap_l_sqr1addc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       a*a+c+d[0] is calculated, where a,c and d[0] are words.
%       The LSW of the result replace d[0] and the MSW of the result
%       the return value.
%
%
%       Word    ap_l_sqr1addc(c,d,a)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of source/destination
```



```
%       Word    a       // input word
%
%
% Entry:
%
c       IS      $0      % carry
d       IS      $1      % pointer to the LSW of the destination/source
a       IS      $2      % operand
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       12 instruction
%
%       c=3     d[0]=1  a=2
%       12,2,21;0,0
%
%



%
        PREFIX  :ap_l_sqr2:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (a0+a1*B)^2 is calculated, where B=2^64
%
%
%       Word    ap_l_sqr2(d,a0,a1)
%
%       Word    *d      // pointer to the LSW of source/destination
%       Word    a0      // input LSW
%       Word    a1      // input MSW
%
%
% Entry:
%
d       IS      $0      % pointer to the LSW of the destination
a0      IS      $1      % operand LSW
a1      IS      $2      % operand MSW
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       25 instructions
%
%       a0=2    a1=3
%       25,3,52;0,0
%
%



%
        PREFIX  :ap_l_sqr2addcc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       (a0+a1*B)^2+c[0]+c[1]*B+d[0]+d[1]*B is calculated, where B=2^64
%
%
%       Word    ap_l_sqr2addcc(c,d,a0,a1)
%
%       Word    *c      // pointer to the two-word in/out carry
```

```
%       Word    *d      // pointer to the LSW of source/destination
%       Word    a0      // input LSW
%       Word    a1      // input MSW
%
%
% Entry:
%
c       IS      $0      % pointer to the LSW of the carry
d       IS      $1      % pointer to the LSW of the destination/source
a0      IS      $2      % operand LSW
a1      IS      $3      % operand MSW
%
% Exit: c[0], c[1] set
%
%
% Profile:
%
%       44 instructions
%
%       c[0]=1 c[1]=2 d[0]=1001 d[1]=1002 a0=2 a1=3
%       44,8,71;0,0
%
%



%
        PREFIX  :ap_l_sqr_class:
%
%       (third-rank candidate to rewrite in assembly)
%
%       This routine squares an arbitrary precision natural number
%       pointed by s. The numbers has length l, where
%       l>0 is supposed. The least significant 2*l-1 word of
%       the result is put to the place pointed by d, the MSW is the
%       return value.
%
%
%       Word    ap_l_sqr_class(d,s,l)
%
%       Word    *c      // pointer to the LSW of destination
%       Word    *d      // pointer to the LSW of source
%       Size    l       // length of source in words, always positive
%
%
% Entry:
%
d       IS      $0      % pointer to LSW of destination
s       IS      $1      % pointer to LSW of source
l       IS      $2      % length of source in words, always positive
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       162 instructions
%
%       treshold=2
%
%       l=1
%       12,2,23;1,1
%
%       l=2
%       34,5,63;2,1
%
%       l=3
%       108,23,170;3,4
%
```



```
%       l=4
%       155,31,255;3,5
%
%       l=5
%       244,49,389;9,5
%
%       l=6
%       309,57,510;9,6
%
%       l=7
%       428,81,692;17,6
%
%       l=8
%       511,89,849;17,7
%
%       l=9
%       660,119,1079;27,7
%
%       l=10
%       761,127,1272;27,8
%
%       l=11
%       940,163,1550;39,8
%
%       l=12
%       1059,171,1779;39,9
%
%       l=13
%       1268,213,2105;53,9
%
%       l=14
%       1406,221,2370;53,10
%
%       l=15
%       1644,269,2744;69,10
%
%       l=16
%       1799,277,3045;69,11
%
%


%
        PREFIX  :ap_l_mul11subc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       d[0]-c-a*b is calculated
%
%
%       Word    ap_l_mul11subc(d,s,a,m)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of source/destination
%       Word    a       // multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % carry
d       IS      $1      % pointer to the LSW of the destination/source
a       IS      $2      % multiplicand
m       IS      $3      % multiplier
%
%
% Exit: MSW of the result; i.e. -2^64*MSW+d[0] is the result
%
%
% Profile:
```

```
%
%       12 instructions
%
%       c=4     d[0]=1  a=2     m=3
%       12,2,21;0,0
%
%



%
        PREFIX  :ap_l_mul21subc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       d[0]+d[1]*B-(s[0]+s[1]*B)*m-c is calculated, where B=2^64
%
%
%       Word    ap_l_mul21subc(c,d,s,m)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m       // multiplier
%
%
% Entry:
%
c       IS      $0      % carry
d       IS      $1      % pointer to the LSW of the destination/source
s       IS      $2      % pointer to the LSW of multiplicand
m       IS      $3      % multiplier
%
%
% Exit: MSW of the result; i.e. -2^128*MSW+d[0]+d[1]*B is the result
%
%
% Profile:
%
%       26 instructions
%
%       c=4 d[0]=1 d[1]=2 s[0]=1001 s[0]=1002 m=3
%       26,6,44;0,0
%
%



%
        PREFIX  :ap_l_mul22subcc:
%
%       (first-rank candidate to rewrite in assembly)
%
%       d[0]+d[1])*B-(s[0]+s[1]*B)*(m0+m1*B)-c[0]-c[1]*B is calculated,
%       where B=2^64
%
%
%       Word    ap_l_mul22subcc(c,d,s,m0,m1)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m0      // multiplier, LSW
%       Word    m1      // multiplier, LSW
%
%
% Entry:
%
c       IS      $0      % pointer to the LSW of the carries c[0], c[1]
d       IS      $1      % pointer to the LSW of the destination/source
s       IS      $2      % pointer to the LSW of multiplicand
```



```
m0      IS      $3      % multiplier, LSW
m1      IS      $4      % multiplier, MSW
%
%
% Exit: c[0], c[1] are set; i.e., the result is
% (c[0]+c[1]*B)*B^2-d[0]-d[1]*B
%
%
% Profile:
%
%       48 instructions
%
%       c[0]=2001 c[1]=2002 d[0]=1 d[1]=2 s[0]=1001 s[1]=1002 m0=3 m1=4
%       48,10,84;0,0
%
%



%
        PREFIX  :ap_l_mul1subc:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a word m and adds a carry word c to the
%       product. The result is added to the destination pointed by d.
%       The length of d and s is l. The new carry is returned.
%
%
%       Word    ap_l_mul1subc(c,d,s,m0,m1)
%
%       Word    c       // carry
%       Word    *d      // pointer to the LSW of source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m       // multiplier
%       Size    l       // length of source and destination in words, >0
%
%
% Entry:
%
c       IS      $0      % carry word
d       IS      $1      % pointer to LSW of destination
s       IS      $2      % pointer to LSW of source
m       IS      $3      % multiplier
l       IS      $4      % length of source and destination in words, >0
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%       20 instructions
%
%       c=0,    m=0
%
%       l=1
%       20,3,31;0,1
%
%       l=2
%       34,6,54;1,1
%
%       l=5
%       76,15,123;4,1
%
%       l=10
%       146,30,238;9,1
%
%       l=20
```

```
%       286,60,468;19,1
%
%       l=50
%       706,150,1158;49,1
%
%       l=100
%       1406,300,2306;99,1
%
%       l=200
%       2806,600,4606;199,1
%
%       l=500
%       7006,1500,11506;499,1
%
%
%       c=0,    m=#ffffffffffffffff
%
%       l=1
%       20,3,31;0,1
%
%       l=2
%       34,6,54;1,1
%
%       l=5
%       76,15,123;4,1
%
%       l=10
%       146,30,238;9,1
%
%       l=20
%       286,60,468;19,1
%
%       l=50
%       706,150,1158;49,1
%
%       l=100
%       1406,300,2306;99,1
%
%       l=200
%       2806,600,4606;199,1
%
%       l=500
%       7006,1500,11506;499,1
%
%



%
        PREFIX  :ap_l_mul2sub:
%
%       (second-rank candidate to rewrite in assembly)
%
%       This routine multiplies an arbitrary precision natural number
%       pointed by s with a two-word number m0+m1*B, where
%       B=2^64, and adds a carry word c.
%       The result is stored at place pointed by d.
%       The length of d and s is l>1.
%       The MSW is returned, the second MSW is stored.
%
%
%       Word    ap_l_mul2sub(d,s,m0,m1,l)
%
%       Word    *d      // pointer to the LSW of source/destination
%       Word    *s      // pointer to the LSW of multiplicand
%       Word    m0      // multiplier, LSW
%       Word    m1      // multiplier, MSW
%       Size    l       // length of source and destination in words, >0
%
%
```



```
% Entry:
%
d       IS      $0      % pointer to LSW of destination
s       IS      $1      % pointer to LSW of source
m0      IS      $2      % multiplier
m1      IS      $3      % multiplier
l       IS      $4      % length of source and destination in words
%
%
% Exit: MSW of the result
%
%
% Profile:
%
%
%       47 instructions
%
%       m0=3    m1=4    s[0]=1 ...      d[1]=1001 ...
%
%       l=2
%       47,7,85;0,1
%
%       l=5
%       116,16,208;3,1
%
%       l=10
%       231,31,413;8,1
%
%       l=20
%       461,61,823;18,1
%
%       l=50
%       1151,151,2053;48,1
%
%       l=100
%       2301,301,4103;98,1
%
%       l=200
%       4601,601,8203;198,1
%
%       l=500
%       11501,1501,20503;498,1
%
%
%       m0=3    m1=#ffffffffffffffff    s[0]=1 ...      d[1]=1001 ...
%
%       l=2
%       47,7,85;0,1
%
%       l=5
%       116,16,208;3,1
%
%       l=10
%       231,31,413;8,1
%
%       l=20
%       461,61,823;18,1
%
%       l=50
%       1151,151,2053;48,1
%
%       l=100
%       2301,301,4103;98,1
%
%       l=200
%       4601,601,8203;198,1
%
%       l=500
%       11501,1501,20503;498,1
```

```
%
%



%
        PREFIX  :ap_l_div21:
%
%       (first-rank candidate to rewrite in assembly)
%
%       The quotient (a[0]+ah*B)/b and the remainder are calculated; B=2^64.
%       We suppose that b1<>0, and the quotient is one word.
%
%
%       Word    ap_l_div21(ah,a,b)
%
%       Word    ah      // MSW of dividend
%       Word    *a      // pointer to the LSW of source/destination
%       Word    b       // divisor
%
%
% Entry:
%
ah      IS      $0      % MSW of dividend
a       IS      $1      % pointer to the LSW of source/destination
b       IS      $2      % divisor
%
%
% Exit: quotient; remainder in a[0]
%
%
% Profile:
%
%       5 instructions
%
%       b=3     a[0]=1  ah=2
%       5,2,64;0,0
%
%



%
        PREFIX  :ap_l_div21r:
%
%       (first-rank candidate to rewrite in assembly)
%
%       the quotient (a[0]+ah*B)/b and the remainder are calculated; B=2^64.
%       WE SUPPOSE THAT b>=B/2 and the quotient is one word.
%
%
%       Word    ap_l_div21r(ah,a,b,rec)
%
%       Word    ah      // MSW of dividend
%       Word    *a      // pointer to the LSW of source/destination
%       Word    b       // divisor, b>=B/2=2^63
%       Word    rec     // reciprocial: floor((2^128-1)/b-B)
%
%
% Entry:
%
ah      IS      $0      % MSW of dividend
a       IS      $1      % pointer to the LSW of source/destination
b       IS      $2      % divisor, b>=B/2=2^63
rec     IS      $3      % reciprocial: floor((2^128-1)/b-B)
%
%
% Exit: quotient; remainder in a[0]
%
%
% Profile:
```



```
%
%       30 instructions
%
%       b=#c000000000000000     a[0]=1  ah=2    rec=#5555555555555555
%       30,2,48;0,0
%
%
%



%
        PREFIX  :quo32:
%
%       (first-rank candidate to rewrite in assembly)
%
%       The quotient (as[0]+as[1]*B+as[2]*B^2)/(b0+b1*B)
%       is calculated, where the dividend is
%       (a[-2]+a[-1]*B+a[0]*B^2+ah*B^3)>>(64-lz), and  B=2^64.
%       We suppose that b1>=B/2, and the quotient is two words.
%
%
%       Word    quo32(ah,a,b,lz)
%
%       Word    ah      // MSW of unshifted dividend
%       Word    *a      // pointer to the second MSW of the unshifted dividend
%       Word    *b      // pointer to the LSW of the shifted divisor
%       Word    lz      // leading zeros of the unshifted divisor
%
%
% Entry:
%
ah      IS      $0      % MSW of unshifted dividend
a       IS      $1      % pointer to the second MSW of the unshifted dividend
b       IS      $2      % pointer to the LSW of the shifted divisor
lz      IS      $3      % leading zeros of the unshifted divisor
%
%
% Exit: quotient MSW
%
%
% Profile:
%
%       52 instructions
%
%       b[1]=#c000000000000000 b[0]=0
%       ah=2 a[0]=3 a[-1]=2 a[-2]=1 lz=62
%       40,7,117;1,0
%
%



%
        PREFIX  :quo32r:
%
%       (first-rank candidate to rewrite in assembly)
%
%       The quotient (as[0]+as[1]*B+as[2]*B^2)/(b0+b1*B)
%       is calculated, where the dividend is
%       (a[-2]+a[-1]*B+a[0]*B^2+ah*B^3)>>(64-lz), and  B=2^64.
%       WE SUPPOSE THAT b1>=B/2, and the quotient is two words.
%       Reciprocal floor((2^128-1)/b1-B) is used.
%
%
%       Word    quo32r(ah,a,b,lz,rec)
%
%       Word    ah      // MSW of unshifted dividend
%       Word    *a      // pointer to the second MSW of the unshifted dividend
%       Word    *b      // pointer to the LSW of the shifted divisor
%       Word    lz      // leading zeros of the unshifted divisor
%       Word    rec     // reciprocial: floor((2^128-1)/b1-B),
```

```
%                           where b1>=B/2=2^63
%
% Entry:
%
ah      IS      $0      % MSW of unshifted dividend
a       IS      $1      % pointer to the second MSW of the unshifted dividend
b       IS      $2      % pointer to the LSW of the shifted divisor
lz      IS      $3      % leading zeros of the unshifted divisor
rec     IS      $4      % reciprocial: floor((2^128-1)/b1-B),
%                         where b1>=B/2=2^63
%
% Exit: quotient MSW
%
%
% Profile:
%
%       76 instructions
%
%       b[1]=#c000000000000000 b[0]=0 lz=62 rec=#5555555555555555
%       ah=2 a[0]=3 a[-1]=2 a[-2]=1
%       64,6,103;0,1
%
%



%
        PREFIX  :ap_l_div42:
%
%       (second-rank candidate to rewrite in assembly)
%
%       the quotient (a[0]+a[1]*B+a[2]*B^2+ah*B^3)/(b0+b1*B)
%       and the remainder are calculated; B=2^64.
%       We suppose that b1*B+b0<>0, and the quotient is two words.
%
%
%       Word    ap_l_div42(ah,a,b0,b1)
%
%       Word    ah      // MSW of dividend
%       Word    *a      // pointer to the LSW of source/destination
%       Word    b0      // divisor LSW
%       Word    b1      // divisor MSW
%
% Entry:
%
ah      IS      $0      % MSW of dividend
a       IS      $1      % pointer to the LSW of source/destination
b0      IS      $2      % divisor LSW
b1      IS      $3      % divisor MSW
%
% Exit: quotient MSW; remainder in a[0], a[1]; quotient LSW in a[2]
%
%
% Profile:
%
%
%       120 instructions
%
%       ah=0 a[0]=1 a[1]=2 a[2]=3 b0=2 b1=3
%       73,6,234;1,2
%
%



%
        PREFIX  :ap_l_div42r:
%
%       (second-rank candidate to rewrite in assembly)
%
%       The quotient (a[0]+a[1]*B+a[2]*B^2+ah*B^3)/(b0+b1*B)
```



```
%       and the remainder are calculated; B=2^64.
%       WE SUPPOSE THAT b1>=B/2, and the quotient is two words.
%
%
%       Word    ap_l_div42r(ah,a,b0,b1,rec)
%
%       Word    ah      // MSW of unshifted dividend
%       Word    *a      // pointer to the second MSW of the unshifted dividend
%       Word    b0      // LSW of the divisor
%       Word    b1      // MSW of the divisor
%       Word    rec     // reciprocial: floor((2^128-1)/b1-B),
%                          where b1>=B/2=2^63
%
%
% Entry:
%
ah      IS      $0      % MSW of dividend
a       IS      $1      % pointer to the LSW of source/destination
b0      IS      $2      % divisor LSW
b1      IS      $3      % divisor MSW
rec     IS      $4      % reciprocial: floor((2^128-1)/b1-B),
%                         where b1>=B/2=2^63
%
%
% Exit: quotient MSW; remainder in a[0], a[1]; quotient LSW in a[2]
%
%
% Profile:
%
%       simple version:
%
%       140 instructions
%
%       ah=0 a[0]=1 a[1]=2 a[2]=3
%       b0=2 b1=#c000000000000000 rec=5555555555555555
%       102,6,178;0,2
%
%



%
        PREFIX  :ap_l_div42rr:
%
%       (second-rank candidate to rewrite in assembly)
%
%       The quotient (a[0]+a[1]*B+a[2]*B^2+ah*B^3)/(b0+b1*B)
%       and the remainder are calculated; B=2^64.
%       WE SUPPOSE THAT b1>=B/2, and the quotient is two words.
%
%
%       Word    ap_l_div42rr(ah,a,b0,b1,rec0,rec1)
%
%       Word    ah      // MSW of unshifted dividend
%       Word    *a      // pointer to the second MSW of the unshifted dividend
%       Word    b0      // LSW of the divisor
%       Word    b1      // MSW of the divisor
%       Word    rec0    // reciprocial LSW
%       Word    rec1    // reciprocial MSW:
%
%       rec0+rec1*B=floor((2^256-1)/(b1*B+b0)-B^2), where b1>=B/2=2^63
%
%
% Entry:
%
ah      IS      $0      % MSW of dividend
a       IS      $1      % pointer to the LSW of source/destination
b0      IS      $2      % divisor LSW
b1      IS      $3      % divisor MSW
rec0    IS      $4      % reciprocial LSW
```

```
rec1    IS      $5      % reciprocial MSW:
%
%       rec0+rec1*B=floor((2^256-1)/(b1*B+b0)-B^2), where b1>=B/2=2^63
%
% Exit: quotient MSW; remainder in a[0], a[1]; quotient LSW in a[2]
%
%
% Profile:
%
%       102 instructions
%
%       ah=0 a[0]=1 a[1]=2 a[2]=3 b0=0 b1=#c000000000000000
%       rec0=#5555555555555555 rec1=#5555555555555555
%       104,6,176;0,0
%
%



%
        PREFIX  :ap_l_div1:
%
%
%       (second-rank candidate to rewrite in assembly)
%
%       The quotient r[]/d is calculated, where d<>0.
%       If quotient not needed, set q to pointer to r[1]
%
%
%       Word    ap_l_div1(q,r,l,d)
%
%       Word    *q      // pointer to the LSW of the quotient
%       Word    *r      // pointer to the LSW of the remainder/dividend
%       Size    l       // size of the dividend, l>=1
%       Word    d       // divisor
%
%
% Entry:
%
q       IS      $0      % pointer to the LSW of the quotient
r       IS      $1      % pointer to the LSW of the remainder/dividend
l       IS      $2      % size of the dividend, l>=1
d       IS      $3      % divisor
%
% Exit: quotient MSW; other words of quotient at array q
%
%
% Profile:
%
%       102 instructions
%
%       d=3     r[0]=1 r[1]=2 ...
%
%       l=1
%       11,2,15;1,2
%
%       l=2
%       18,4,81;2,2
%
%       l=5
%       191,10,348;5,2
%
%       l=10
%       367,20,613;10,2
%
%       l=20
%       717,40,1143;20,2
%
%       l=50
%       1767,100,2733;50,2
```



```
%
%       l=100
%       3517,200,5383;100,2
%
%       l=200
%       7017,400,10683;200,2
%
%       l=500
%       17517,1000,26583;500,2
%
%



%
        PREFIX  :ap_l_div2:
%
%       (second-rank candidate to rewrite in assembly)
%
%       The quotient r[]/(dl+dh*B) is calculated,
%       where B=2^64 and dh<>0.
%       If quotient not needed, set q to the address of r[2].
%
%
%       Word    ap_l_div2(q,r,l,dh,dl)
%
%       Word    *q      // pointer to the LSW of the quotient
%       Word    *r      // pointer to the LSW of the remainder/dividend
%       Size    l       // size of the dividend, l>=1
%       Word    dh      // MSW of the divisor
%       Word    dl      // LSW of the divisor
%
%
% Entry:
%
q       IS      $0      % pointer to the LSW of the quotient
r       IS      $1      % pointer to the LSW of the remainder/dividend
l       IS      $2      % size of the dividend, l>=2
dh      IS      $3      % MSW of the divisor
dl      IS      $4      % LSW of the divisor
%
% Exit: quotient MSW; other words of quotient at array q
%
%
% Profile:
%
%       279 instructions
%
%       dh=3 dl=2 r[0]=1 r[1]=2 ...
%
%       l=2
%       25,4,32;1,2
%
%       l=5
%       165,10,482;9,3
%
%       l=10
%       590,20,990;18,7
%
%       l=20
%       1160,40,1920;38,7
%
%       l=50
%       3035,100,4905;98,22
%
%       l=100
%       6149,200,9867;198,46
%
%       l=200
%       12388,400,19804;398,95
```

```
%
%       l=500
%       31160,1000,49680;998,247
%
%



%
        PREFIX  :ap_l_div_class:
%
%       (third-rank candidate to rewrite in assembly)
%
%       The quotient d[]/s[] and the remainder are calculated
%       If quotient not needed, set q point to r[ls]
%
%
%       Word    ap_l_div_class(q,r,l,dh,dl)
%
%       Word    *q      // pointer to the LSW of the quotient
%       Word    *r      // pointer to the LSW of the remainder/dividend
%       Size    ld      // size of the dividend
%       Word    *s      // pointer to the LSW of the source/divisor
%       Size    ls      // size of the divisor, ld>=ls>0
%
%
% Entry:
%
q       IS      $0      % pointer to the LSW of the quotient
r       IS      $1      % pointer to the LSW of the remainder/dividend
ld      IS      $2      % size of the dividend
s       IS      $3      % pointer to the LSW of the source/divisor
ls      IS      $4      % size of the divisor, ld>=ls>0
%
% Exit: quotient MSW; other words of quotient in array q
%
%
% Profile:
%
%       583 instructions
%
%       r[0]=1001 r[1]=1002 ... s[0]=1 s[1]=2 ...
%
%       ls=5    ld=5
%       126,15,255;8,2
%
%       ls=5    ld=6
%       222,23,475;13,3
%
%       ls=5    ld=10
%       825,55,1401;29,14
%
%       ls=10   ld=10
%       196,30,384;13,2
%
%       ls=10   ld=11
%       362,53,705;23,3
%
%       ls=10   ld=20
%       2325,260,3835;103,31
%
%       ls=20   ld=20
%       336,60,600;23,2
%
%       ls=20   ld=21
%       642,113,1165;43,3
%
%       ls=20   ld=40
%       7345,1120,12057;405,57
%
```



```
%       ls=50   ld=50
%       756,150,1290;53,2
%
%       ls=50   ld=51
%       1484,293,2545;103,3
%
%       ls=50   ld=100
%       41049,7864,66163;2700,133
%
%       ls=100  ld=100
%       1456,300,2440;103,2
%
%       ls=100  ld=101
%       2882,593,4845;203,3
%
%       ls=100  ld=200
%       154101,31346,249229;10603,265
%
%       ls=200  ld=200
%       2856,600,4740;203,2
%
%       ls=200  ld=201
%       5682,1193,9445;403,3
%
%       ls=200  ld=400
%       603769,127474,973865;42812,499
%
%       ls=500  ld=500
%       7056,1500,11640;503,2
%
%       ls=500  ld=501
%       14084,2993,23245;1003,3
%
%       ls=500  ld=1000
%       3832739,835969,6107675;279492,1169
%
%



%
%



%       KARASTUBA'S METHOD



%
        PREFIX  :kara:
%
%       kara(d,s1,s2,b1,b2,n) (fourth-rank candidate to rewrite in assembly)
%
%
%       The following routine do Karatsubas multiplication
%       recursively.
%
% Entry: $0=d; $1=s1; $2=s2; $3=b1; $4=b2; $5=n
%       where d,s1,s2 are destination and source pointers,
%       respectively, b1 and b2 are buffer pointers.
%       b1=s1 and/or b2=s2 possible, in which case the
%       corresponding source(s) will be destroyed. Here n is
%       the lenght of the sources and the buffers.
%       The length of the destination is 2n.
%       Recursive halfing and calls are used until the length
%       became smaller then the treshold constant.
%
% Exit: -
%
%
% Profile:
```

```
%
%       Not yet written
%


%
        PREFIX  :kara_short:
%
%       kara_short(d,s1,s2,n)
%
%       (first-rank candidate to rewrite in assembly)
%
%

%       This routine works as kara, but
%       does everything in registers.  On MMIX, it works for 1<=n<=7,
%       i.e. on MMIX the treshold is planned to be 8.
%
% Entry: $0=d; $1=s1; $2=s2; $3=n
%       where d,s1,s2 are destination and source pointers,
%       respectively, and n is
%       the lenght of the sources and the buffers.
%       The length of the destination is 2n.
%
% Exit: -
%
%
% Profile:
%
% l=    1
%       9,4,22;0,1
%
% l=    2
%       36,8,76;1,1
%
% l=    3
%       84,12,169;2,1
%
% l=    4
%       151,16,299;3,1
%
% l=    5
%       239,20,468;4,1
%
% l=    6
%       346,24,672;6,0
%
% l=    7
%       472,28,917;5,1
%
%
%
%
        PREFIX  :kara_sqr:
%
%       kara_sqr(d,s,b,n)
%
%       (fourth-rank candidate to rewrite in assembly)
%
%
%
%       The following routine do Karatsubas squaring
%       recursively.
%
% Entry: $0=d; $1=s; $2=b; $3=n
%       where d,s are destination and source pointers,
%       respectively, b is a buffer pointer.
%       b=s possible, in which case the
%       source will be destroyed. Here n is
%       the lenght of the source and the buffer.
%       The length of the destination is 2n.
%       Recursive halfing and calls are used until the length
%       became smaller then the treshold constant.
```



```
%
% Exit: -
%
% Profile:
%
%       Not yet written
%


%
        PREFIX  :kara_sqr_short:
%
%       (first-rank candidate to rewrite in assembly)
%
%       This routine works as kara_sqr, but
%       does everything in registers.  On MMIX, it works for 1<=n<=7,
%       i.e. on MMIX the treshold is planned to be 8.
%
%
% Entry:
%
%
d       IS      $0      % pointer to the LSW of the destination
s       IS      $1      % pointer to the LSW of the source
l       IS      $2      % length of sources in words, always positive
%
%
% Exit: -
%
%
% Profile:
%
% l=    1
%       7,2,20;0,1
%
% l=    2
%       33,5,64;1,1
%
% l=    3
%       66,8,124;2,1
%
% l=    4
%       109,11,203;3,1
%
% l=    5
%       162,14,301;4,1
%
% l=    6
%       225,17,416;6,0
%
% l=    7
%       296,20,552;5,1
%



%       FOURIER TRANSFORMATION OVER THE COMPLEX NUMBERS




%       SPARSE NUMBERS




%       FOURIER TRANSFORMATION MODULO A FERMAT NUMBER
```

```
%       FAST NATURAL NUMBER ARITHMETIC



%       MODULAR ARITHMETIC
```