Description of problem

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

pe001.s

```
.syntax unified

.equ max3start,999
.equ max5start,995

number  .req r4
matched .req r5
sum     .req r6
max3    .req r7
max5    .req r8

.section .rodata
        .align  2
string:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r8, lr}
        ldr     max5,   =max5start
        ldr     max3,   =max3start
        ldr     number, =max3start    @ start at 1000 - 1 ; numbers < 1000
        mov     matched, 0
        mov     sum, 0
loop:
        cmp     number, max3
        bne     test5

# matched a multiple of 3 - decrement max3, add to sum and set matched to 1
        mov     matched, 1
        add     sum, sum, number
        subs    max3, max3, 3

test5:
        cmp     number, max5
        bne     last

# matched a multiple of 5 - decrement max5, add to sum and set matched to 1
        subs    max5, max5, 5
        cmp     matched, 1              @ have we already added it?
        addne   sum, sum, number        @ if not add it to the total

last:
# decrement number and reset matched and loop
        mov     matched, 0
        subs    number, number, 1
        bne     loop

        mov     r1, sum
        ldr     r0, =string     @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r8, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

pe002.s

```
.syntax unified

.equ maxfib,4000000

previous        .req r4
current         .req r5
next            .req r6
sum             .req r7
max             .req r8
tmp             .req r9

.section .rodata
        .align  2
sumstring:
        .asciz "%d\n"

.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r9, lr}
        ldr     max,   =maxfib
        mov     previous, 1
        mov     current, 1
        mov     sum, 0

loop:
        cmp     current, max
        bgt     last

        add     next, current, previous

        movs    tmp, current, lsr 1 @ set carry flag from lsr - for the odd-valued terms
        addcc   sum, sum, current   @ these are even-valued fibonacci (when cc is true)
```

```
        mov     previous, current
        mov     current, next
        b       loop

last:
        mov     r1, sum
        ldr     r0, =sumstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r9, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143 ?

pe003.s

```
.syntax unified

@ 600851475143 is 8BE589EAC7 in hex

.equ numhi,139        @ 0x8b
.equ numlo,3851020999 @ 0xe589eac7

num_hi  .req r4
num_lo  .req r5
maxdiv  .req r6
n       .req r7

.section .rodata
 .align 2
resstring:
 .asciz "%d\n"
.text
 .align 2
 .global main
 .type main, %function
main:
 stmfd sp!, {r4-r7, lr}

 mov maxdiv, 0
 mov n, 1
 ldr num_lo, =numlo
 ldr num_hi, =numhi
loop:
 add n, n, 2           @ start at 3 and increment by 2
loop1:
 mov r0, num_lo
 mov r1, num_hi
 mov r2, n
 mov r3, 0
 bl  long_divide
 teq r2, 0
 bne loop
 teq r3, 0
 bne loop
 mov num_lo, r0         @ only get here when we have no remainder
 mov num_hi, r1         @ save the divisor as the new number
 mov r0, n
 bl isprime
 teq r0, 1
 bne loop               @ increment n if n is non-prime
 cmp maxdiv, n
 movlt maxdiv, n        @ save n as the largest divisor if it is larger
 teq num_lo, 1          @ we know it has prime factors
 beq printme
 b loop1
printme:
 mov r1, maxdiv
 ldr r0, =resstring     @ store address of start of string to r0
 bl printf

 mov r0, 0
 ldmfd sp!, {r4-r7, pc}
 mov r7, 1              @ set r7 to 1 - the syscall for exit
 swi 0                  @ then invoke the syscall from linux
```

isprime.s

```
.syntax unified

# this subroutine returns 1 if the passed number is prime; 0 if not

# inputs
#   r0 - integer to test

# outputs
#   r0 - prime boolean

number          .req r4
divisor         .req r5
tmp             .req r6

.global isprime
.type isprime, %function
.text
.align  2

isprime:
        stmfd   sp!, {r4-r6, lr}
        mov     number, r0
        ands    tmp, number, 1
        bne     odd
        mov     r0, 0
        cmp     number, 2       @ 2 is the only prime even number
        bne     last
        mov     r0, 1
```

```
        b       last
odd:
        mov     divisor, 3
        cmp     number, 8
        bgt     big
        mov     r0, 1
        cmp     number, 1       @ 1 is the only odd number < 8 not prime
        bne     last
        mov     r0, 0
        b       last
big:
        mov     r0, number
        mov     r1, divisor
        bl      divide
        teq     r1, 0
        beq     factor
        add     divisor, divisor, 2
        mul     tmp, divisor, divisor
        subs    tmp, tmp, number
        ble     big
        mov     r0, 1
        b       last
factor:
        mov     r0, 0
last:
        ldmfd   sp!, {r4-r6, pc}
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

long_divide.s

```
.syntax unified
# long_divide takes low numerator in r0, high numerator in r1, low denominator in r2 and high denominator in r3
# returns low quotient in r0, high quotient in r1, low remainder in r2 and high remainder in r3
.global long_divide
.type   long_divide, %function
long_divide:
        stmfd   sp!, {lr}
# see https://codereview.chromium.org/5302007/diff/12001/arch/arm/lib/_uldivmod.S
        bl      __aeabi_uldivmod
        ldmfd   sp!, {pc}
```

Description of problem

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 Ã— 99.

Find the largest palindrome made from the product of two 3-digit numbers.

pe004.s

```
.syntax unified

.equ max3,999
.equ min3,100
.equ maxdigits,6

i       .req r4
j       .req r5
product .req r6
maxp    .req r7
mini    .req r8
minj    .req r9
maxj    .req r10

.section .rodata
        .align  2
sumstring:
        .asciz "%d\n"

.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r10, lr}
        ldr     i, =max3
        ldr     mini, =min3
        ldr     maxj, =max3
        ldr     minj, =min3
iloop:
        mov     j, maxj
jloop:
        mul     product, i, j

        mov     r0, product
        bl      is_palindromic
        cmp     r0, #1
        bne     next
        cmp     product, maxp
        ble     next
        mov     maxp, product
        mov     r0, product
        bl      divide_by_10 @ divides r0 by 10
        bl      divide_by_10 @ so 3 consecutive calls
        bl      divide_by_10 @ will divide by 1000
        mov     minj, r0
        mov     minj, r0

next:
        subs    j, j, 1
```

```
        cmp     j, minj
        bgt     jloop

        subs    i, i, 1
        mov     maxj, i
        cmp     i, mini
        bgt     iloop

last:
        mov     r1, maxp
        ldr     r0, =sumstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

ispalindromic.s

```
.syntax unified

.equ datum_size, 1
.equ digits, 6

# this subroutine returns 1 if the passed 6-digit number is palindromic; 0 if not
# the number is a product of 2 3-digit numbers so we assume the product has 6 digits
#
# inputs
#   r0 - integer to test
#
# outputs
#   r0 - palindromic boolean
#
# local
#
        left            .req r4
        right           .req r5
        counter         .req r6
        buffer_address  .req r7
        running         .req r8
        tmp             .req r9
#
.section .bss
   .lcomm buffer, 6
#
.global is_palindromic
.type is_palindromic, %function
.global get_digits
.type get_digits, %function
.text
.align  2
#
is_palindromic:
        stmfd   sp!, {r4-r9, lr}
        bl      get_digits
        mov     counter, 3
        ldr     buffer_address, =buffer
ip_last:
        sub     left, buffer_address, counter
        ldrb    tmp, [left, 3]
        add     right, buffer_address, counter
        ldrb    running, [right, 2]
        teq     tmp, running
        bne     no
        subs    counter, counter, 1
        bgt     ip_last
        mov     r0, 1
        b       last
no:
        mov     r0, 0
last:
        ldmfd   sp!, {r4-r9, pc}
#
get_digits:
        stmfd   sp!, {r7-r8, lr}
        ldr     running, =digits
        ldr     buffer_address, =buffer
gd_loop:
        subs    running, running, 1
        bl      divide_by_10_remainder
        strb    r1, [buffer_address], #datum_size
        bgt     gd_loop

gd_last:
        ldmfd   sp!, {r7-r8, pc}
```

divide_by_10.s

```
.syntax unified

# this subroutine divides the passed number by 10 and
# returns the dividend and remainder
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xccccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
# and subtract from the original number to give the remainder on division
# by 10.
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend
#   r1 - the remainder

.equ const,-0x33333333
#.equ const,0xccccccccd
.text
.align  2
```

```
.global divide_by_10_remainder
.type   divide_by_10_remainder, %function
divide_by_10_remainder:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     rsmall
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r3, r2          @ r3 = r2
        mov     r3, r3, asl #2  @ r3 = 4 * r3
        add     r3, r3, r2      @ r3 = r3 + r2
        mov     r3, r3, asl #1  @ r3 = 2 * r3
        rsb     r3, r3, r0      @ r3 = r0 - r3 = r0 - 10*int(r0/10)
        mov     r1, r3          @ the remainder
        mov     r0, r2          @ the dividend
        b       rlast
rsmall:
        mov     r1, r0
        mov     r0, 0
rlast:
        ldmfd   sp!, {pc}


# this subroutine divides the passed number by 10
# returns the dividend
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xcccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend

.align  2
.global divide_by_10
.type   divide_by_10, %function
divide_by_10:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     small
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r0, r2          @ the dividend
        b       last
small:
        mov     r0, 0
last:
        ldmfd   sp!, {pc}
```

Description of problem

2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder.

What is the smallest positive number that is *evenly divisible* by all of the numbers from 1 to 20?

pe005.s

```
.syntax unified

.equ    limit,20

.align 4

@ algorithm
@ initialise try_products to 1
@ foreach number > 1 and <= limit
@ test if it is prime
@ if try_products is set, then multiply the number by itself
@ while it does not exceed limit, then multiply the total by
@ this product. if the number squared exceeds the limit, then
@ set try_product to 0.
@ if try_products is 0 and the number is prime, then multiply
@ the total by number.

try_product     .req r4
number          .req r5
last            .req r6
total           .req r7
tmp             .req r8

.section .rodata
        .align  2
resstring:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r8, lr}

        mov     total, 1
        mov     try_product, 1
        mov     number, 2
loop:
        mov     r0, number
        bl      isprime20
        cmp     r0, 1
        bne     nexti

        cmp     try_product, 1
        bne     no_product
        mul     tmp, number, number
        cmp     tmp, limit
        ble     prod_start
        mov     try_product, 0
        b       no_product
prod_start:
```

```
        mov     tmp, number
        mov     last, tmp
prod_loop:
        cmp     tmp, limit
        bgt     last_mul
        mov     last, tmp
        mul     tmp, tmp, number
        b       prod_loop
last_mul:
        mul     total, total, last
no_product:
        cmp     try_product, 0
        bne     nexti
        mul     total, total, number
nexti:
        cmp     number, limit
        beq     printme
        add     number, number, 1
        b       loop

printme:
        mov     r1, total
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r8, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

# this subroutine returns 1 if the passed number (<= 20) is prime; 0 if not
#
# inputs
#   r0 - integer to test
#
# outputs
#   r0 - prime boolean

.global isprime20
.type isprime20, %function
.text
.align  2

isprime20:
        stmfd   sp!, {lr}
        mov     r1, r0
        ands    r2, r1, 1
        bne     odd
        mov     r0, 0
        cmp     r1, 2   @ 2 is the only prime even r1
        bne     last
        mov     r0, 1
        b       last
odd:
        mov     r0, 1
        cmp     r1, 9
        bne     test15
        mov     r0, 0
        b       last
test15:
        cmp     r1, 15
        bne     last
        mov     r0, 0
last:
        ldmfd   sp!, {pc}
```

Description of problem

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + ... + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + ... + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is $3025 - 385 = 2640$.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

pe006.s

```
.syntax unified

.equ limit,100

number  .req r4
sumsq   .req r5
sqsum   .req r6
tmp     .req r7

.section .rodata
        .align  2
string:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r7, lr}
        mov     sqsum, 0
        mov     sumsq, 0
        ldr     number, =limit
loop:
        mul     tmp, number, number
        add     sqsum, sqsum, tmp
# decrement number and loop or exit
        subs    number, number, 1
        beq     end_loop
        b       loop
end_loop:
        ldr     number, =limit
        add     number, number, 1
```

```
        ldr     sumsq, =limit
        mul     sumsq, sumsq, number
        lsr     sumsq, sumsq, 1
        mul     sumsq, sumsq, sumsq
last:
        sub     tmp, sumsq, sqsum
        mov     r1, tmp
        ldr     r0, =string     @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r7, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13, we can see that the 6th prime is 13.

What is the 10 001st prime number?

pe007.s

```
.syntax unified

.equ    limit,10000
.equ    limit4,40000

.align 4

number          .req r4
count           .req r5
numprimes       .req r6
primes_ptr      .req r7

.section .bss
.lcomm primes_vector,limit4

.section .rodata
        .align  2
resstring:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r7, lr}

        ldr     primes_ptr, =primes_vector
        mov     numprimes, 1
        mov     number, 2
        str     number, [primes_ptr]

        ldr     count, =limit
        mov     number, 3       @ 2 is the first prime
loop:
        mov     r0, number
        ldr     r1, =primes_vector
        mov     r2, numprimes
        bl      prime_vector
        teq     r0, 1
        bne     nexti
        str     number, [primes_ptr, numprimes, lsl 2]
        add     numprimes, numprimes, 1
        subs    count, count, 1
        beq     printme
nexti:
        add     number, number, 2
        b       loop

printme:
        mov     r1, number
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r7, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

prime_vector.s

```
.syntax unified
.equ    word, 4

# this subroutine returns 1 if the passed number is prime; 0 if not
#
# inputs
#   r0 - integer to test
#   r1 - pointer to vector of prime integers smaller than r0
#   r2 - length of vector passed in r1
#
# outputs
#   r0 - prime boolean

number          .req r4
vptr            .req r5
tmp             .req r6
squared         .req r7
vsize           .req r8

.global prime_vector
.type prime_vector, %function
.text
.align  2

prime_vector:
        stmfd   sp!, {r4-r8, lr}
        mov     number, r0
        mov     vptr, r1
        mov     vsize, r2
nexti:
```

```
        ldr     tmp, [vptr], word
        mul     squared, tmp, tmp
        cmp     squared, number
        movgt   r0, 1
        bgt     last
        mov     r0, number
        mov     r1, tmp
        bl      divide
        teq     r1, 0
        moveq   r0, 0
        beq     last
        subs    vsize, vsize, 1
        bgt     nexti
        mov     r0, 1
last:
        ldmfd   sp!, {r4-r8, pc}
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

The four adjacent digits in the 1000-digit number that have the greatest product are 9 Ã— 9 Ã— 8 Ã— 9 = 5832.

73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450

Find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product?

pe008.s

```
.syntax unified

.equ    limit,10000
.equ    outer, 988
.equ    inner, 13
.equ    address_offset, 12 @inner - 1

.align 4

address         .req r4
thisbyte        .req r5
icounter        .req r6
ocounter        .req r7
maxv_lo         .req r8
maxv_hi         .req r9
tmp_lo          .req r10
tmp_hi          .req r11
carry           .req r12
addoff          .req r12
tmp             .req r0

.section .data
buffer:
.byte 7, 3, 1, 6, 7, 1, 7, 6, 5, 3, 1, 3, 3, 0, 6, 2, 4, 9, 1, 9, 2, 2
.byte 5, 1, 1, 9, 6, 7, 4, 4, 2, 6, 5, 7, 4, 7, 4, 2, 3, 5, 5, 3, 4, 9, 1, 9
.byte 4, 9, 3, 4, 9, 6, 9, 8, 3, 5, 2, 0, 3, 1, 2, 7, 7, 4, 5, 0, 6, 3, 2, 6
.byte 2, 3, 9, 5, 7, 8, 3, 1, 8, 0, 1, 6, 9, 8, 4, 8, 0, 1, 8, 6, 9, 4, 7, 8
.byte 8, 5, 1, 8, 4, 3, 8, 5, 8, 6, 1, 5, 6, 0, 7, 8, 9, 1, 1, 2, 9, 4, 9, 4
.byte 9, 5, 4, 5, 9, 5, 0, 1, 7, 3, 7, 9, 5, 8, 3, 3, 1, 9, 5, 2, 8, 5, 3, 2
.byte 0, 8, 8, 0, 5, 5, 1, 1, 1, 2, 5, 4, 0, 6, 9, 8, 7, 4, 7, 1, 5, 8, 5, 2
.byte 3, 8, 6, 3, 0, 5, 0, 7, 1, 5, 6, 9, 3, 2, 9, 0, 9, 6, 3, 2, 9, 5, 2, 2
.byte 7, 4, 4, 3, 0, 4, 3, 5, 5, 7, 6, 6, 8, 9, 6, 6, 4, 8, 9, 5, 0, 4, 4, 5
.byte 2, 4, 4, 5, 2, 3, 1, 6, 1, 7, 3, 1, 8, 5, 6, 4, 0, 3, 0, 9, 8, 7, 1, 1
.byte 1, 2, 1, 7, 2, 2, 3, 8, 3, 1, 1, 3, 6, 2, 2, 2, 9, 8, 9, 3, 4, 2, 3, 3
.byte 8, 0, 3, 0, 8, 1, 3, 5, 3, 3, 6, 2, 7, 6, 6, 1, 4, 2, 8, 2, 8, 0, 6, 4
.byte 4, 4, 4, 8, 6, 6, 4, 5, 2, 3, 8, 7, 4, 9, 3, 0, 3, 5, 8, 9, 0, 7, 2, 9
.byte 6, 2, 9, 0, 4, 9, 1, 5, 6, 0, 4, 4, 0, 7, 7, 2, 3, 9, 0, 7, 1, 3, 8, 1
.byte 0, 5, 1, 5, 8, 5, 9, 3, 0, 7, 9, 6, 0, 8, 6, 6, 7, 0, 1, 7, 2, 4, 2, 7
.byte 1, 2, 1, 8, 8, 3, 9, 9, 8, 7, 9, 7, 9, 0, 8, 7, 9, 2, 2, 7, 4, 9, 2, 1
.byte 9, 0, 1, 6, 9, 9, 7, 2, 0, 8, 8, 8, 0, 9, 3, 7, 7, 6, 6, 5, 7, 2, 7, 3
.byte 3, 3, 0, 0, 1, 0, 5, 3, 3, 6, 7, 8, 8, 1, 2, 2, 0, 2, 3, 5, 4, 2, 1, 8
.byte 0, 9, 7, 5, 1, 2, 5, 4, 5, 4, 0, 5, 9, 4, 7, 5, 2, 2, 4, 3, 5, 2, 5, 8
.byte 4, 9, 0, 7, 7, 1, 1, 6, 7, 0, 5, 5, 6, 0, 1, 3, 6, 0, 4, 8, 3, 9, 5, 8
.byte 6, 4, 4, 6, 7, 0, 6, 3, 2, 4, 4, 1, 5, 7, 2, 2, 1, 5, 5, 3, 9, 7, 5, 3
.byte 6, 9, 7, 8, 1, 7, 9, 7, 7, 8, 4, 6, 1, 7, 4, 0, 6, 4, 9, 5, 5, 1, 4, 9
.byte 2, 9, 0, 8, 6, 2, 5, 6, 9, 3, 2, 1, 9, 7, 8, 4, 6, 8, 6, 2, 2, 4, 8, 2
.byte 8, 3, 9, 7, 2, 2, 4, 1, 3, 7, 5, 6, 5, 7, 0, 5, 6, 0, 5, 7, 4, 9, 0, 2
.byte 6, 1, 4, 0, 7, 9, 7, 2, 9, 6, 8, 6, 5, 2, 4, 1, 4, 5, 3, 5, 1, 0, 0, 4
.byte 7, 4, 8, 2, 1, 6, 6, 3, 7, 0, 4, 8, 4, 4, 0, 3, 1, 9, 9, 8, 9, 0, 0, 0
```

```
.byte 8, 8, 9, 5, 2, 4, 3, 4, 5, 0, 6, 5, 8, 5, 4, 1, 2, 2, 7, 5, 8, 8, 6, 6
.byte 6, 8, 8, 1, 1, 6, 4, 2, 7, 1, 7, 1, 4, 7, 9, 9, 2, 4, 4, 4, 2, 9, 2, 8
.byte 2, 3, 0, 8, 6, 3, 4, 6, 5, 6, 7, 4, 8, 1, 3, 9, 1, 9, 1, 2, 3, 1, 6, 2
.byte 8, 2, 4, 5, 8, 6, 1, 7, 8, 6, 6, 4, 5, 8, 3, 5, 9, 1, 2, 4, 5, 6, 6, 5
.byte 2, 9, 4, 7, 6, 5, 4, 5, 6, 8, 2, 8, 4, 8, 9, 1, 2, 8, 8, 3, 1, 4, 2, 6
.byte 0, 7, 6, 9, 0, 0, 4, 2, 2, 4, 2, 1, 9, 0, 2, 2, 6, 7, 1, 0, 5, 5, 6, 2
.byte 6, 3, 2, 1, 1, 1, 1, 1, 0, 9, 3, 7, 0, 5, 4, 4, 2, 1, 7, 5, 0, 6, 9, 4
.byte 1, 6, 5, 8, 9, 6, 0, 4, 0, 8, 0, 7, 1, 9, 8, 4, 0, 3, 8, 5, 0, 9, 6, 2
.byte 4, 5, 5, 4, 4, 4, 3, 6, 2, 9, 8, 1, 2, 3, 0, 9, 8, 7, 8, 7, 9, 9, 2, 7
.byte 2, 4, 4, 2, 8, 4, 9, 0, 9, 1, 8, 8, 8, 4, 5, 8, 0, 1, 5, 6, 1, 6, 6, 0
.byte 9, 7, 9, 1, 9, 1, 3, 3, 8, 7, 5, 4, 9, 9, 2, 0, 0, 5, 2, 4, 0, 6, 3, 6
.byte 8, 9, 9, 1, 2, 5, 6, 0, 7, 1, 7, 6, 0, 6, 0, 5, 8, 8, 6, 1, 1, 6, 4, 6
.byte 7, 1, 0, 9, 4, 0, 5, 0, 7, 7, 5, 4, 1, 0, 0, 2, 2, 5, 6, 9, 8, 3, 1, 5
.byte 5, 2, 0, 0, 0, 5, 5, 9, 3, 5, 7, 2, 9, 7, 2, 5, 7, 1, 6, 3, 6, 2, 6, 9
.byte 5, 6, 1, 8, 8, 2, 6, 7, 0, 4, 2, 8, 2, 5, 2, 4, 8, 3, 6, 0, 0, 8, 2, 3
.byte 2, 5, 7, 5, 3, 0, 4, 2, 0, 7, 5, 2, 9, 6, 3, 4, 5, 0

.section .rodata
        .align  2
llustring:
        .asciz "%llu\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r12, lr}
        mov     maxv_lo, #0
        mov     maxv_hi, #0
        ldr     address, =buffer
        ldr     ocounter, =outer
outer_start:
        ldr     icounter, =inner
        mov     tmp_lo, #1
        mov     tmp_hi, #0
inner_start:
        ldrb    thisbyte, [address], 1
        umull   tmp_lo, carry, tmp_lo, thisbyte @ multiply 64 bit tmp
        mla     tmp_hi, tmp_hi, thisbyte, carry @ by thisbyte
        subs    icounter, icounter, 1
        bne     inner_start
        cmp     maxv_lo, tmp_lo                 @ compare 2 64 bit numbers
        sbcs    tmp, maxv_hi, tmp_hi            @ low then high halves
        movlt   maxv_lo, tmp_lo
        movlt   maxv_hi, tmp_hi
        ldr     addoff, =address_offset
        sub     address, address, addoff
        subs    ocounter, ocounter, 1
        bne     outer_start

printme:
        mov     r2, maxv_lo
        mov     r3, maxv_hi
        ldr     r0, =llustring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r12, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

A Pythagorean triplet is a set of three natural numbers, $a < b < c$, for which,

$a^2 + b^2 = c^2$

For example, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

There exists exactly one Pythagorean triplet for which $a + b + c = 1000$.
Find the product abc.

pe009.s

```
.syntax unified

.equ    limit,1000

.align 4

icount  .req r4
jcount  .req r5
kcount  .req r6
tmp     .req r8
jksum   .req r9

.section .rodata
        .align  2
resstring:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r9, lr}
        ldr     icount, =limit
istart:
        subs    jcount, icount, 1
        beq     nexti
jstart:
        subs    kcount, jcount, 1
        beq     nextj
        add     tmp, icount, jcount
kstart:
        add     tmp, tmp, kcount
        cmp     tmp, #limit
        bne     nextk
        mul     jksum, kcount, kcount
        mul     tmp, jcount, jcount
        add     jksum, jksum, tmp
        mul     tmp, icount, icount
        cmp     jksum, tmp
```

```
        beq     printme
nextk:
        add     tmp, icount, jcount
        subs    kcount, kcount, 1
        bne     kstart
nextj:
        subs    jcount, jcount, 1
        bne     jstart
nexti:
        subs    icount, icount, 1
        bne     istart
printme:
        mul     tmp, icount, jcount
        mul     tmp, tmp, kcount
        mov     r1, tmp
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r9, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

The sum of the primes below 10 is 2 + 3 + 5 + 7 = 17.

Find the sum of all the primes below two million.

pe010.s

```
.syntax unified
.equ    word,4
.equ    logword,2

.equ    limit,2000000
.equ    numprimes4,595732

sum_hi          .req r4
sum_lo          .req r5
numprimes       .req r6
primes_ptr      .req r7
number          .req r8
limit           .req r9

.align 2

.section .bss
.lcomm primes_vector,numprimes4

.section .rodata
        .align  2
llustring:
        .asciz "%llu\n"

.text
.align  8
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r9, fp, lr}

        ldr     primes_ptr, =primes_vector
        mov     numprimes, 1
        mov     number, 2
        strb    number, [primes_ptr]
        ldr     limit, =limit
        mov     sum_hi, 0
        mov     sum_lo, 2
        mov     number, 3
loop:
        cmp     number, limit
        bge     printme

        mov     r0, number
        ldr     r1, =primes_vector
        mov     r2, numprimes
        bl      prime_vector
        teq     r0, 1
        bne     nexti
        str     number, [primes_ptr, numprimes, lsl 2]
        add     numprimes, numprimes, 1
        adds    sum_lo, sum_lo, number
        adc     sum_hi, sum_hi, 0
nexti:
        add     number, number, 2
        b       loop
printme:
        mov     r2, sum_lo
        mov     r3, sum_hi
        ldr     r0, =llustring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r9, fp, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

prime_vector.s

```
.syntax unified
.equ    word, 4

# this subroutine returns 1 if the passed number is prime; 0 if not
#
# inputs
#   r0 - integer to test
#   r1 - pointer to vector of prime integers smaller than r0
#   r2 - length of vector passed in r1
#
# outputs
#   r0 - prime boolean
```

```
number          .req r4
vptr            .req r5
tmp             .req r6
squared         .req r7
vsize           .req r8

.global prime_vector
.type prime_vector, %function
.text
.align  2

prime_vector:
        stmfd   sp!, {r4-r8, lr}
        mov     number, r0
        mov     vptr, r1
        mov     vsize, r2
nexti:
        ldr     tmp, [vptr], word
        mul     squared, tmp, tmp
        cmp     squared, number
        movgt   r0, 1
        bgt     last
        mov     r0, number
        mov     r1, tmp
        bl      divide
        teq     r1, 0
        moveq   r0, 0
        beq     last
        subs    vsize, vsize, 1
        bgt     nexti
        mov     r0, 1
last:
        ldmfd   sp!, {r4-r8, pc}
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

In the 20Ã—20 grid below, four numbers along a diagonal line have been marked in red.

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 **26** 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 **63** 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 **78** 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 **14** 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

The product of these numbers is 26 Ã— 63 Ã— 78 Ã— 14 = 1788696.

What is the greatest product of four adjacent numbers in the same direction (up, down, left, right, or diagonally) in the 20Ã—20 grid?

pe011.s

```
.syntax unified

.align 2

.section .rodata
buffer:
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.byte 0, 0, 0, 8, 2, 22, 97, 38, 15, 0, 40, 0, 75, 4, 5, 7, 78, 52, 12, 50, 77, 91, 8, 0, 0, 0
.byte 0, 0, 0, 49, 49, 99, 40, 17, 81, 18, 57, 60, 87, 17, 40, 98, 43, 69, 48, 4, 56, 62, 0, 0, 0, 0
.byte 0, 0, 0, 81, 49, 31, 73, 55, 79, 14, 29, 93, 71, 40, 67, 53, 88, 30, 3, 49, 13, 36, 65, 0, 0, 0
.byte 0, 0, 0, 52, 70, 95, 23, 4, 60, 11, 42, 69, 24, 68, 56, 1, 32, 56, 71, 37, 2, 36, 91, 0, 0, 0
.byte 0, 0, 0, 22, 31, 16, 71, 51, 67, 63, 89, 41, 92, 36, 54, 22, 40, 40, 28, 66, 33, 13, 80, 0, 0, 0
.byte 0, 0, 0, 24, 47, 32, 60, 99, 3, 45, 2, 44, 75, 33, 53, 78, 36, 84, 20, 35, 17, 12, 50, 0, 0, 0
.byte 0, 0, 0, 32, 98, 81, 28, 64, 23, 67, 10, 26, 38, 40, 67, 59, 54, 70, 66, 18, 38, 64, 70, 0, 0, 0
.byte 0, 0, 0, 67, 26, 20, 68, 2, 62, 12, 20, 95, 63, 94, 39, 63, 8, 40, 91, 66, 49, 94, 21, 0, 0, 0
.byte 0, 0, 0, 24, 55, 58, 5, 66, 73, 99, 26, 97, 17, 78, 78, 96, 83, 14, 88, 34, 89, 63, 72, 0, 0, 0
.byte 0, 0, 0, 21, 36, 23, 9, 75, 0, 76, 44, 20, 45, 35, 14, 0, 61, 33, 97, 34, 31, 33, 95, 0, 0, 0
.byte 0, 0, 0, 78, 17, 53, 28, 22, 75, 31, 67, 15, 94, 3, 80, 4, 62, 16, 14, 9, 53, 56, 92, 0, 0, 0
.byte 0, 0, 0, 16, 39, 5, 42, 96, 35, 31, 47, 55, 58, 88, 24, 0, 17, 54, 24, 36, 29, 85, 57, 0, 0, 0
.byte 0, 0, 0, 86, 56, 0, 48, 35, 71, 89, 7, 5, 44, 44, 37, 44, 60, 21, 58, 51, 54, 17, 58, 0, 0, 0
.byte 0, 0, 0, 19, 80, 81, 68, 5, 94, 47, 69, 28, 73, 92, 13, 86, 52, 17, 77, 4, 89, 55, 40, 0, 0, 0
.byte 0, 0, 0, 4, 52, 8, 83, 97, 35, 99, 16, 7, 97, 57, 32, 16, 26, 26, 79, 33, 27, 98, 66, 0, 0, 0
.byte 0, 0, 0, 88, 36, 68, 87, 57, 62, 20, 72, 3, 46, 33, 67, 46, 55, 12, 32, 63, 93, 53, 69, 0, 0, 0
.byte 0, 0, 0, 4, 42, 16, 73, 38, 25, 39, 11, 24, 94, 72, 18, 8, 46, 29, 32, 40, 62, 76, 36, 0, 0, 0
.byte 0, 0, 0, 20, 69, 36, 41, 72, 30, 23, 88, 34, 62, 99, 69, 82, 67, 59, 85, 74, 4, 36, 16, 0, 0, 0
.byte 0, 0, 0, 20, 73, 35, 29, 78, 31, 90, 1, 74, 31, 49, 71, 48, 86, 81, 16, 23, 57, 5, 54, 0, 0, 0
```

```
.byte 0, 0, 0, 1, 70, 54, 71, 83, 51, 54, 69, 16, 92, 33, 48, 61, 43, 52, 1, 89, 19, 67, 48, 0, 0, 0
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

resstring:
        .asciz "%d\n"

.equ    ocolumns, 26
.equ    icolumns, 20
.equ    colstart, 23
.equ    border, 3
.equ    offset, 2
.equ    points, 4
.equ    north, -ocolumns
.equ    north_east, -ocolumns+1
.equ    east, 1
.equ    south_east, ocolumns+1
.equ    south, ocolumns
.equ    south_west, ocolumns - 1
.equ    west, -1
.equ    north_west, -ocolumns-1
.equ    point_set, 1

tmp             .req r2
pointv          .req r3
data_ptr        .req r4
icount          .req r5
jcount          .req r6
kcount          .req r7
maxv            .req r8
dpstore         .req r9

.macro direction a
        mov     data_ptr, dpstore
        mov     kcount, points
        mov     pointv, point_set
kloop\@:
        ldrb    tmp, [data_ptr], \a
        mul     pointv, pointv, tmp
        subs    kcount, kcount, 1
        bne     kloop\@
        cmp     maxv, pointv
        movlt   maxv, pointv    @ set maxv to max of maxv and pointv
.endm

.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r9, lr}
        mov     icount, icolumns
iloop:
        mov     jcount, icolumns
jloop:
        mov     tmp, ocolumns
        add     data_ptr, icount, offset
        mul     tmp, tmp, data_ptr

        add     tmp, tmp, jcount
        add     tmp, tmp, offset
        ldr     data_ptr, =buffer
        add     dpstore, data_ptr, tmp  @ data[i][j]

        direction north
        direction north_east
        direction east
        direction south_east
        direction south
        direction south_west
        direction west
        direction north_west

        subs    jcount, jcount, 1
        bne     jloop

        subs    icount, icount, 1
        bne     iloop
printme:
        mov     r1, maxv
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r9, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. The first ten terms would be:

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Let us list the factors of the first seven triangle numbers:

1: 1
3: 1,3
6: 1,2,3,6
10: 1,2,5,10
15: 1,3,5,15
21: 1,3,7,21
28: 1,2,4,7,14,28

We can see that 28 is the first triangle number to have over five divisors.

What is the value of the first triangle number to have over five hundred divisors?

pe012.s

```
.syntax unified
```

```
.align 2

.section .rodata
resstring:
        .asciz "%d\n"

.equ    last, 250

icount          .req r4
jcount          .req r5
tmp             .req r6
num             .req r7

.text
        .align  2
        .global get_num_divisors
        .type   get_num_divisors, %function
get_num_divisors:
        stmfd   sp!, {r4-r7, lr}
        mov     num, r0
        mov     icount, 0
        mov     jcount, 1
jstart:
        mov     r0, num
        mov     r1, jcount
        bl      divide
        teq     r1, 0
        beq     factor
        b       nextj
factor:
        add     icount, icount, 1
nextj:
        add     jcount, jcount, 1
        mul     tmp, jcount, jcount
        cmp     num, tmp
        bgt     jstart
        mov     r0, icount
        ldmfd   sp!, {r4-r7, pc}

        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r7, lr}
        mov     num, 0
        mov     icount, 0
        mov     jcount, 1
loop:
        cmp     num, #last
        bge     printme
        add     icount, icount, jcount
        mov     r0, icount
        bl      get_num_divisors
        mov     num, r0

        add     jcount, jcount, 1
        b       loop

printme:
        mov     r1, icount
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r7, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.

37107287533902102798797998220837590246510135740250
46376937677490009712648124896970078050417018260538
74324986199524741059474233309513058123726617309629
91942213363574161572522430563301811072406154908250
23067588207539346171171980310421047513778063246676
89261670696623633820136378418383684178734361726757
28112879812849979408065481931592621691275889832738
44274228917432520321923589422876796487670272189318
47451445736001306439091167216856844588711603153276
70386486105843025439939619828917593665686757934951
62176457141856560629502157223196586755079324193331
64906352462741904929101432445813822663347944758178
92575867718337217661963751590579239728245598838407
58203565325359399008402633568948830189458628227828
80181199384826282014278194139940567587151170094390
35398664372827112653829987240784473053190104293586
86515506006295864861532075273371959191420517255829
71693888707715466499115593487603532921714970056938
54370070576826684624621495650076471787294438377604
53282654108756828443191190634694037855217779295145
36123272525000296071075082563815656710885258350721
45876576172410976447339110607218265236877223636045
17423706905851860660448207621209813287860733969412
81142660418086830619328460811191061556940512689692

```
51934325451728388641918047049293215058642563049483
62467221648435076201727918039944693004732956340691
15732444386908125794514089057706229429197107928209
55037687525678773091862540744969844508330393682126
18336384825330154686196124348767681297534375946515
80386287592878490201521685554828717201219257766954
78182833757993103614740356856449095527097864797581
16726320100436897842553539920931837441497806860984
48403098129077791799088218795327364475675590848030
87086987551392711854517078544161852424320693150332
59959406895756536782107074926966537676326235447210
69793950679652694742597709739166693763042633987085
41052684708299085211399427365734116182760315001271
65378607361501080857009149939512557028198746004375
35829035317434717326932123578154982629742552737307
94953759765105305946966067683156574377167401875275
88902802571733229619176668713819931811048770190271
25267680276078003013678680992525463401061632866526
36270218540497705585629946580636237993140746255962
24074486908231174977792365466257246923322810917141
91430288197103288597806669760892938638285025333403
34413065578016127815921815005561868836468420090470
23053081172816430487623791969842487255036638784583
11487696932154902810424020138335124462181441773470
63783299490636259666498587618221225225512486764533
67720186971698544312419572409913959008952310058822
95548255300263520781532296796249481641953868218774
76085327132285723110424803456124867697064507995236
37774242535411291684276865538926205024910326572967
23701913275725675285653248258265463092207058596522
29798860272258331913126375147341994889534765745501
18495701454879288984856827726077713721403798879715
38298203783031473527721580348144513491373226651381
34829543829199918180278916522431027392251122869539
40957953066405232632538044100059654939159879593635
29746152185502371307642255121183693803580388584903
41698116222072977186158236678424689157993532961922
62467957194401269043877107275048102390895523597457
23189706772547915061505504953922979530901129967519
86188088225875314529584099251203829009407770775672
11306739708304724483816533873502340845647058077308
82959174767140363198008187129011875491310547126581
97623331044818386269515456334926366572897563400500
42846280183517070527831839425882145521227251250327
55121603546981200581762165212827652751691296897789
32238195734329339946437501907836945765883352399886
75506164965184775180738168837861091527357929701337
62177842752192623401942399639168044983993173312731
32924185707147349566916674687634660915035914677504
99518671430235219628894890102423325116913619626622
73267460800591547471830798392868535206946944540724
76841822524674417161514036427982273348055556214818
97142617910342598647204516893989422179826088076852
87783646182799346313767754307809363333018982642090
10848802521674670883215120185883543223812876952786
71329612474782464538636993009049310363619763878039
62184073572399794223406235393808339651327408011116
66627891981488087797941876876144230030984490851411
60661826293682836764744779239180335110989069790714
85786944089552990653640447425576083659976645795096
66024396409905389607120198219976047599490197230297
64913982680032973156037120041377903785566085089252
16730939319872750275468906903707539413042652315011
94809377245048795150954100921645863754710598436791
78639167021187492431995700641917969777599028300699
15368713711936614952811305876380278410754449733078
40789923115535562561142322423255033685442488917353
44889911501440648020369068063960672322193204149535
41503128880339536053299340368006977710650566631954
81234880673210146739058568557934581403627822703280
82616570773948327592232845941706525094512325230608
22918802058777319719839450180888072429661980811197
77158542502016545090413245809786882778948721859617
72107838435069186155435662884062257473692284509516
20849603980134001723930671666823555245252804609722
53503534226472524250874054075591789781264330331690
```

pe013.s

```
.syntax unified

.equ ten, 10
.equ hundred, 100
.equ col_nums,50
.equ row_nums,100
.equ col_offset,50
@ maximum sum of any column is 900 - if all elements were 9

.align 4

const           .req r1
byte            .req r2
col_num         .req r4
row_num         .req r5
row1            .req r6
data_ptr        .req r8
```

```
col_sum         .req r9
tmp             .req r10

.section .bss
   .lcomm result, 52

.section .data
buffer:
.byte 3, 7, 1, 0, 7, 2, 8, 7, 5, 3, 3, 9, 0, 2, 1, 0, 2, 7, 9, 8, 7, 9, 7, 9, 9, 8, 2, 2, 0, 8, 3, 7, 5, 9, 0, 2, 4, 6, 5, 1, 0, 1, 3, 5, 7, 4, 0, 2, 5, 0
.byte 4, 6, 3, 7, 6, 9, 3, 7, 6, 7, 7, 4, 9, 0, 0, 0, 9, 7, 1, 2, 6, 4, 8, 1, 2, 4, 8, 9, 6, 9, 7, 0, 0, 7, 8, 0, 5, 0, 4, 1, 7, 0, 1, 8, 2, 6, 0, 5, 3, 8
.byte 7, 4, 3, 2, 4, 9, 8, 6, 1, 9, 9, 5, 2, 4, 7, 4, 1, 0, 5, 9, 4, 7, 4, 2, 3, 3, 3, 0, 9, 5, 1, 3, 0, 5, 8, 1, 2, 3, 7, 2, 6, 6, 1, 7, 3, 0, 9, 6, 2, 9
.byte 9, 1, 9, 4, 2, 2, 1, 3, 3, 6, 3, 5, 7, 4, 1, 6, 1, 5, 7, 2, 5, 2, 2, 4, 3, 0, 5, 6, 3, 3, 0, 1, 8, 1, 1, 0, 7, 2, 4, 0, 6, 1, 5, 4, 9, 0, 8, 2, 5, 0
.byte 2, 3, 0, 6, 7, 5, 8, 8, 2, 0, 7, 5, 3, 9, 3, 4, 6, 1, 7, 1, 1, 7, 1, 9, 8, 0, 3, 1, 0, 4, 2, 1, 0, 4, 7, 5, 1, 3, 7, 7, 8, 0, 6, 3, 2, 4, 6, 6, 7, 6
.byte 8, 9, 2, 6, 1, 6, 7, 0, 6, 9, 6, 6, 2, 3, 6, 3, 3, 8, 2, 0, 1, 3, 6, 3, 7, 8, 4, 1, 8, 3, 8, 3, 6, 8, 4, 1, 7, 8, 7, 3, 4, 3, 6, 1, 7, 2, 6, 7, 5, 7
.byte 2, 8, 1, 1, 2, 8, 7, 9, 8, 1, 2, 8, 4, 9, 9, 7, 9, 4, 0, 8, 0, 6, 5, 4, 8, 1, 9, 3, 1, 5, 9, 2, 6, 2, 1, 6, 9, 1, 2, 7, 5, 8, 8, 9, 8, 3, 2, 7, 3, 8
.byte 4, 4, 2, 7, 4, 2, 2, 8, 9, 1, 7, 4, 3, 2, 5, 2, 0, 3, 2, 1, 9, 2, 3, 5, 8, 9, 4, 2, 2, 8, 7, 6, 7, 9, 6, 4, 8, 7, 6, 7, 0, 2, 7, 2, 1, 8, 9, 3, 1, 8
.byte 4, 7, 4, 5, 1, 4, 4, 5, 7, 3, 6, 0, 0, 1, 3, 0, 6, 4, 3, 9, 0, 9, 1, 1, 6, 7, 2, 1, 6, 8, 5, 6, 8, 4, 4, 5, 8, 8, 7, 1, 1, 6, 0, 3, 1, 5, 3, 2, 7, 6
.byte 7, 0, 3, 8, 6, 4, 8, 6, 1, 0, 5, 8, 4, 3, 0, 2, 5, 4, 3, 9, 9, 3, 9, 6, 1, 9, 8, 2, 8, 9, 1, 7, 5, 9, 3, 6, 6, 5, 6, 8, 6, 7, 5, 7, 9, 3, 4, 9, 5, 1
.byte 6, 2, 1, 7, 6, 4, 5, 7, 1, 4, 1, 8, 5, 6, 5, 6, 0, 6, 2, 9, 5, 0, 2, 1, 5, 7, 2, 2, 3, 1, 9, 6, 5, 8, 6, 7, 5, 5, 0, 7, 9, 3, 2, 4, 1, 9, 3, 3, 3, 1
.byte 6, 4, 9, 0, 6, 3, 5, 2, 4, 6, 2, 7, 4, 1, 9, 0, 4, 9, 2, 9, 1, 0, 1, 4, 3, 2, 4, 4, 5, 8, 1, 3, 8, 2, 2, 6, 6, 3, 3, 4, 7, 9, 4, 4, 7, 5, 8, 1, 7, 8
.byte 9, 2, 5, 7, 5, 8, 6, 7, 7, 1, 8, 3, 3, 7, 2, 1, 7, 6, 6, 1, 9, 6, 3, 7, 5, 1, 5, 9, 0, 5, 7, 9, 2, 3, 9, 7, 2, 8, 2, 4, 5, 5, 9, 8, 8, 3, 8, 4, 0, 7
.byte 5, 8, 2, 0, 3, 5, 6, 5, 3, 2, 5, 3, 5, 9, 3, 9, 9, 0, 0, 8, 4, 0, 2, 6, 3, 3, 5, 6, 8, 9, 4, 8, 8, 3, 0, 1, 8, 9, 4, 5, 8, 6, 2, 8, 2, 2, 7, 8, 2, 8
.byte 8, 0, 1, 8, 1, 1, 9, 9, 3, 8, 4, 8, 2, 6, 2, 8, 2, 0, 1, 4, 2, 7, 8, 1, 9, 4, 1, 3, 9, 9, 4, 0, 5, 6, 7, 5, 8, 7, 1, 5, 1, 1, 7, 0, 0, 9, 4, 3, 9, 0
.byte 3, 5, 3, 9, 8, 6, 6, 4, 3, 7, 2, 8, 2, 7, 1, 1, 2, 6, 5, 3, 8, 2, 9, 9, 8, 7, 2, 4, 0, 7, 8, 4, 4, 7, 3, 0, 5, 3, 1, 9, 0, 1, 0, 4, 2, 9, 3, 5, 8, 6
.byte 8, 6, 5, 1, 5, 5, 0, 6, 0, 0, 6, 2, 9, 5, 8, 6, 4, 8, 6, 1, 5, 3, 2, 0, 7, 5, 2, 7, 3, 3, 7, 1, 9, 5, 9, 1, 9, 1, 4, 2, 0, 5, 1, 7, 2, 5, 5, 8, 2, 9
.byte 7, 1, 6, 9, 3, 8, 8, 8, 7, 0, 7, 7, 1, 5, 4, 6, 6, 4, 9, 9, 1, 1, 5, 5, 9, 3, 4, 8, 7, 6, 0, 3, 5, 3, 2, 9, 2, 1, 7, 1, 4, 9, 7, 0, 0, 5, 6, 9, 3, 8
.byte 5, 4, 3, 7, 0, 0, 7, 0, 5, 7, 6, 8, 2, 6, 6, 8, 4, 6, 2, 4, 6, 2, 1, 4, 9, 5, 6, 5, 0, 0, 7, 6, 4, 7, 1, 7, 8, 7, 2, 9, 4, 4, 3, 8, 3, 7, 7, 6, 0, 4
.byte 5, 3, 2, 8, 2, 6, 5, 4, 1, 0, 8, 7, 5, 6, 8, 2, 8, 4, 4, 3, 1, 9, 1, 1, 9, 0, 6, 3, 4, 6, 9, 4, 0, 3, 7, 8, 5, 5, 2, 1, 7, 7, 7, 9, 2, 9, 5, 1, 4, 5
.byte 3, 6, 1, 2, 3, 2, 7, 2, 5, 2, 5, 0, 0, 0, 2, 9, 6, 0, 7, 1, 0, 7, 5, 0, 8, 2, 5, 6, 3, 8, 1, 5, 6, 5, 6, 7, 1, 0, 8, 8, 5, 2, 5, 8, 3, 5, 0, 7, 2, 1
.byte 4, 5, 8, 7, 6, 5, 7, 6, 1, 7, 2, 4, 1, 0, 9, 7, 6, 4, 4, 7, 3, 3, 9, 1, 1, 0, 6, 0, 7, 2, 1, 8, 2, 6, 5, 2, 3, 6, 8, 7, 7, 2, 2, 3, 6, 3, 6, 0, 4, 5
.byte 1, 7, 4, 2, 3, 7, 0, 6, 9, 0, 5, 8, 5, 1, 8, 6, 0, 6, 6, 0, 4, 4, 8, 2, 0, 7, 6, 2, 1, 2, 0, 9, 8, 1, 3, 2, 8, 7, 8, 6, 0, 7, 3, 3, 9, 6, 9, 4, 1, 2
.byte 8, 1, 1, 4, 2, 6, 6, 0, 4, 1, 8, 0, 8, 6, 8, 3, 0, 6, 1, 9, 3, 2, 8, 4, 6, 0, 8, 1, 1, 1, 9, 1, 0, 6, 1, 5, 5, 6, 9, 4, 0, 5, 1, 2, 6, 8, 9, 6, 9, 2
.byte 5, 1, 9, 3, 4, 3, 2, 5, 4, 5, 1, 7, 2, 8, 3, 8, 8, 6, 4, 1, 9, 1, 8, 0, 4, 7, 0, 4, 9, 2, 9, 3, 2, 1, 5, 0, 5, 8, 6, 4, 2, 5, 6, 3, 0, 4, 9, 4, 8, 3
.byte 6, 2, 4, 6, 7, 2, 2, 1, 6, 4, 8, 4, 3, 5, 0, 7, 6, 2, 0, 1, 7, 2, 7, 9, 1, 8, 0, 3, 9, 9, 4, 4, 6, 9, 3, 0, 0, 4, 7, 3, 2, 9, 5, 6, 3, 4, 0, 6, 9, 1
.byte 1, 5, 7, 3, 2, 4, 4, 4, 3, 8, 6, 9, 0, 8, 1, 2, 5, 7, 9, 4, 5, 1, 4, 0, 8, 9, 0, 5, 7, 7, 0, 6, 2, 2, 9, 4, 2, 9, 1, 9, 7, 1, 0, 7, 9, 2, 8, 2, 0, 9
.byte 5, 5, 0, 3, 7, 6, 8, 7, 5, 2, 5, 6, 7, 8, 7, 7, 3, 0, 9, 1, 8, 6, 2, 5, 4, 0, 7, 4, 4, 9, 6, 9, 8, 4, 4, 5, 0, 8, 3, 3, 0, 3, 9, 3, 6, 8, 2, 1, 2, 6
.byte 1, 8, 3, 3, 6, 3, 8, 4, 8, 2, 5, 3, 3, 0, 1, 5, 4, 6, 8, 6, 1, 9, 6, 1, 2, 4, 3, 4, 8, 7, 6, 7, 6, 8, 1, 2, 9, 7, 5, 3, 4, 3, 7, 5, 9, 4, 6, 5, 1, 5
.byte 8, 0, 3, 8, 6, 2, 8, 7, 5, 9, 2, 8, 7, 8, 4, 9, 0, 2, 0, 1, 5, 2, 1, 6, 8, 5, 5, 5, 4, 8, 2, 8, 7, 1, 7, 2, 0, 1, 2, 1, 9, 2, 5, 7, 7, 6, 6, 9, 5, 4
.byte 7, 8, 1, 8, 2, 8, 3, 3, 7, 5, 7, 9, 9, 3, 1, 0, 3, 6, 1, 4, 7, 4, 0, 3, 5, 6, 8, 5, 6, 4, 4, 9, 0, 9, 5, 5, 2, 7, 0, 9, 7, 8, 6, 4, 7, 9, 7, 5, 8, 1
.byte 1, 6, 7, 2, 6, 3, 2, 0, 1, 0, 0, 4, 3, 6, 8, 9, 7, 8, 4, 2, 5, 5, 3, 5, 3, 9, 9, 2, 0, 9, 3, 1, 8, 3, 7, 4, 4, 1, 4, 9, 7, 8, 0, 6, 8, 6, 0, 9, 8, 4
.byte 4, 8, 4, 0, 3, 0, 9, 8, 1, 2, 9, 0, 7, 7, 7, 9, 1, 7, 9, 9, 0, 8, 8, 2, 1, 8, 7, 9, 5, 3, 2, 7, 3, 6, 4, 4, 7, 5, 6, 7, 5, 5, 9, 0, 8, 4, 8, 0, 3, 0
.byte 8, 7, 0, 8, 6, 9, 8, 7, 5, 5, 1, 3, 9, 2, 7, 1, 1, 8, 5, 4, 5, 1, 7, 0, 7, 8, 5, 4, 4, 1, 6, 1, 8, 5, 2, 4, 2, 4, 3, 2, 0, 6, 9, 3, 1, 5, 0, 3, 3, 2
.byte 5, 9, 9, 5, 9, 4, 0, 6, 8, 9, 5, 7, 5, 6, 5, 3, 6, 7, 8, 2, 1, 0, 7, 0, 7, 4, 9, 2, 6, 9, 6, 6, 5, 3, 7, 6, 7, 6, 3, 2, 6, 2, 3, 5, 4, 4, 7, 2, 1, 0
.byte 6, 9, 7, 9, 3, 9, 5, 0, 6, 7, 9, 6, 5, 2, 6, 9, 4, 7, 4, 2, 5, 9, 7, 7, 0, 9, 7, 3, 9, 1, 6, 6, 6, 9, 3, 7, 6, 3, 0, 4, 2, 6, 3, 3, 9, 8, 7, 0, 8, 5
.byte 4, 1, 0, 5, 2, 6, 8, 4, 7, 0, 8, 2, 9, 9, 0, 8, 5, 2, 1, 1, 3, 9, 9, 4, 2, 7, 3, 6, 5, 7, 3, 4, 1, 1, 6, 1, 8, 2, 7, 6, 0, 3, 1, 5, 0, 0, 1, 2, 7, 1
.byte 6, 5, 3, 7, 8, 6, 0, 7, 3, 6, 1, 5, 0, 1, 0, 8, 0, 8, 5, 7, 0, 0, 9, 1, 4, 9, 9, 3, 9, 5, 1, 2, 5, 5, 7, 0, 2, 8, 1, 9, 8, 7, 4, 6, 0, 0, 4, 3, 7, 5
.byte 3, 5, 8, 2, 9, 0, 3, 5, 3, 1, 7, 4, 3, 4, 7, 1, 7, 3, 2, 6, 9, 3, 2, 1, 2, 3, 5, 7, 8, 1, 5, 4, 9, 8, 2, 6, 2, 9, 7, 4, 2, 5, 5, 2, 7, 3, 7, 3, 0, 7
.byte 9, 4, 9, 5, 3, 7, 5, 9, 7, 6, 5, 1, 0, 5, 3, 0, 5, 9, 4, 6, 9, 6, 6, 0, 6, 7, 6, 8, 3, 1, 5, 6, 5, 7, 4, 3, 7, 7, 1, 6, 7, 4, 0, 1, 8, 7, 5, 2, 7, 5
.byte 8, 8, 9, 0, 2, 8, 0, 2, 5, 7, 1, 7, 3, 3, 2, 2, 9, 6, 1, 9, 1, 7, 6, 6, 6, 8, 7, 1, 3, 8, 1, 9, 9, 3, 1, 8, 1, 1, 0, 4, 8, 7, 7, 0, 1, 9, 0, 2, 7, 1
.byte 2, 5, 2, 6, 7, 6, 8, 0, 2, 7, 6, 0, 7, 8, 0, 0, 3, 0, 1, 3, 6, 7, 8, 6, 8, 0, 9, 9, 2, 5, 2, 5, 4, 6, 3, 4, 0, 1, 0, 6, 1, 6, 3, 2, 8, 6, 6, 5, 2, 6
.byte 3, 6, 2, 7, 0, 2, 1, 8, 5, 4, 0, 4, 9, 7, 7, 0, 5, 5, 8, 5, 6, 2, 9, 9, 4, 6, 5, 8, 0, 6, 3, 6, 2, 3, 7, 9, 9, 3, 1, 4, 0, 7, 4, 6, 2, 5, 5, 9, 6, 2
.byte 2, 4, 0, 7, 4, 4, 8, 6, 9, 0, 8, 2, 3, 1, 1, 7, 4, 9, 7, 7, 7, 9, 2, 3, 6, 5, 4, 6, 6, 2, 5, 7, 2, 4, 6, 9, 2, 3, 3, 2, 2, 8, 1, 0, 9, 1, 7, 1, 4, 1
.byte 9, 1, 4, 3, 0, 2, 8, 8, 1, 9, 7, 1, 0, 3, 2, 8, 8, 5, 9, 7, 8, 0, 6, 6, 6, 9, 7, 6, 0, 8, 9, 2, 9, 3, 8, 6, 3, 8, 2, 8, 5, 0, 2, 5, 3, 3, 3, 4, 0, 3
.byte 3, 4, 4, 1, 3, 0, 6, 5, 5, 7, 8, 0, 1, 6, 1, 2, 7, 8, 1, 5, 9, 2, 1, 8, 1, 5, 0, 0, 5, 5, 6, 1, 8, 6, 8, 8, 3, 6, 4, 6, 8, 4, 2, 0, 0, 9, 0, 4, 7, 0
.byte 2, 3, 0, 5, 3, 0, 8, 1, 1, 7, 2, 8, 1, 6, 4, 3, 0, 4, 8, 7, 6, 2, 3, 7, 9, 1, 9, 6, 9, 8, 4, 2, 4, 8, 7, 2, 5, 5, 0, 3, 6, 6, 3, 8, 7, 8, 4, 5, 8, 3
.byte 1, 1, 4, 8, 7, 6, 9, 6, 9, 3, 2, 1, 5, 4, 9, 0, 2, 8, 1, 0, 4, 2, 4, 0, 2, 0, 1, 3, 8, 3, 3, 5, 1, 2, 4, 4, 6, 2, 1, 8, 1, 4, 4, 1, 7, 7, 3, 4, 7, 0
.byte 6, 3, 7, 8, 3, 2, 9, 9, 4, 9, 0, 6, 3, 6, 2, 5, 9, 6, 6, 6, 4, 9, 8, 5, 8, 7, 6, 1, 8, 2, 2, 1, 2, 2, 5, 2, 2, 5, 5, 1, 2, 4, 8, 6, 7, 6, 4, 5, 3, 3
.byte 6, 7, 7, 2, 0, 1, 8, 6, 9, 7, 1, 6, 9, 8, 5, 4, 4, 3, 1, 2, 4, 1, 9, 5, 7, 2, 4, 0, 9, 9, 1, 3, 9, 5, 9, 0, 0, 8, 9, 5, 2, 3, 1, 0, 0, 5, 8, 8, 2, 2
.byte 9, 5, 5, 4, 8, 2, 5, 5, 3, 0, 0, 2, 6, 3, 5, 2, 0, 7, 8, 1, 5, 3, 2, 2, 9, 6, 7, 9, 6, 2, 4, 9, 4, 8, 1, 6, 4, 1, 9, 5, 3, 8, 6, 8, 2, 1, 8, 7, 7, 4
.byte 7, 6, 0, 8, 5, 3, 2, 7, 1, 3, 2, 2, 8, 5, 7, 2, 3, 1, 1, 0, 4, 2, 4, 8, 0, 3, 4, 5, 6, 1, 2, 4, 8, 6, 7, 6, 9, 7, 0, 6, 4, 5, 0, 7, 9, 9, 5, 2, 3, 6
.byte 3, 7, 7, 7, 4, 2, 4, 2, 5, 3, 5, 4, 1, 1, 2, 9, 1, 6, 8, 4, 2, 7, 6, 8, 6, 5, 5, 3, 8, 9, 2, 6, 2, 0, 5, 0, 2, 4, 9, 1, 0, 3, 2, 6, 5, 7, 2, 9, 6, 7
.byte 2, 3, 7, 0, 1, 9, 1, 3, 2, 7, 5, 7, 2, 5, 6, 7, 5, 2, 8, 5, 6, 5, 3, 2, 4, 8, 2, 5, 8, 2, 6, 5, 4, 6, 3, 0, 9, 2, 2, 0, 7, 0, 5, 8, 5, 9, 6, 5, 2, 2
.byte 2, 9, 7, 9, 8, 8, 6, 0, 2, 7, 2, 2, 5, 8, 3, 3, 1, 9, 1, 3, 1, 2, 6, 3, 7, 5, 1, 4, 7, 3, 4, 1, 9, 9, 4, 8, 8, 9, 5, 3, 4, 7, 6, 5, 7, 4, 5, 5, 0, 1
.byte 1, 8, 4, 9, 5, 7, 0, 1, 4, 5, 4, 8, 7, 9, 2, 8, 8, 9, 8, 4, 8, 5, 6, 8, 2, 7, 7, 2, 6, 0, 7, 7, 7, 1, 3, 7, 2, 1, 4, 0, 3, 7, 9, 8, 8, 7, 9, 7, 1, 5
.byte 3, 8, 2, 9, 8, 2, 0, 3, 7, 8, 3, 0, 3, 1, 4, 7, 3, 5, 2, 7, 7, 2, 1, 5, 8, 0, 3, 4, 8, 1, 4, 4, 5, 1, 3, 4, 9, 1, 3, 7, 3, 2, 2, 6, 6, 5, 1, 3, 8, 1
.byte 3, 4, 8, 2, 9, 5, 4, 3, 8, 2, 9, 1, 9, 9, 9, 1, 8, 1, 8, 0, 2, 7, 8, 9, 1, 6, 5, 2, 2, 4, 3, 1, 0, 2, 7, 3, 9, 2, 2, 5, 1, 1, 2, 2, 8, 6, 9, 5, 3, 9
.byte 4, 0, 9, 5, 7, 9, 5, 3, 0, 6, 6, 4, 0, 5, 2, 3, 2, 6, 3, 2, 5, 3, 8, 0, 4, 4, 1, 0, 0, 0, 5, 9, 6, 5, 4, 9, 3, 9, 1, 5, 9, 8, 7, 9, 5, 9, 3, 6, 3, 5
.byte 2, 9, 7, 4, 6, 1, 5, 2, 1, 8, 5, 5, 0, 2, 3, 7, 1, 3, 0, 7, 6, 4, 2, 2, 5, 5, 1, 2, 1, 1, 8, 3, 6, 9, 3, 8, 0, 3, 5, 8, 0, 3, 8, 8, 5, 8, 4, 9, 0, 3
.byte 4, 1, 6, 9, 8, 1, 1, 6, 2, 2, 2, 0, 7, 2, 9, 7, 7, 1, 8, 6, 1, 5, 8, 2, 3, 6, 6, 7, 8, 4, 2, 4, 6, 8, 9, 1, 5, 7, 9, 9, 3, 5, 3, 2, 9, 6, 1, 9, 2, 2
.byte 6, 2, 4, 6, 7, 9, 5, 7, 1, 9, 4, 4, 0, 1, 2, 6, 9, 0, 4, 3, 8, 7, 7, 1, 0, 7, 2, 7, 5, 0, 4, 8, 1, 0, 2, 3, 9, 0, 8, 9, 5, 5, 2, 3, 5, 9, 7, 4, 5, 7
.byte 2, 3, 1, 8, 9, 7, 0, 6, 7, 7, 2, 5, 4, 7, 9, 1, 5, 0, 6, 1, 5, 0, 5, 5, 0, 4, 9, 5, 3, 9, 2, 2, 9, 7, 9, 5, 3, 0, 9, 0, 1, 1, 2, 9, 9, 6, 7, 5, 1, 9
.byte 8, 6, 1, 8, 8, 0, 8, 8, 2, 2, 5, 8, 7, 5, 3, 1, 4, 5, 2, 9, 5, 8, 4, 0, 9, 9, 2, 5, 1, 2, 0, 3, 8, 2, 9, 0, 0, 9, 4, 0, 7, 7, 7, 0, 7, 7, 5, 6, 7, 2
.byte 1, 1, 3, 0, 6, 7, 3, 9, 7, 0, 8, 3, 0, 4, 7, 2, 4, 4, 8, 3, 8, 1, 6, 5, 3, 3, 8, 7, 3, 5, 0, 2, 3, 4, 0, 8, 4, 5, 6, 4, 7, 0, 5, 8, 0, 7, 7, 3, 0, 8
.byte 8, 2, 9, 5, 9, 1, 7, 4, 7, 6, 7, 1, 4, 0, 3, 6, 3, 1, 9, 8, 0, 0, 8, 1, 8, 7, 1, 2, 9, 0, 1, 1, 8, 7, 5, 4, 9, 1, 3, 1, 0, 5, 4, 7, 1, 2, 6, 5, 8, 1
.byte 9, 7, 6, 2, 3, 3, 3, 1, 0, 4, 4, 8, 1, 8, 3, 8, 6, 2, 6, 9, 5, 1, 5, 4, 5, 6, 3, 3, 4, 9, 2, 6, 3, 6, 6, 5, 7, 2, 8, 9, 7, 5, 6, 3, 4, 0, 0, 5, 0, 0
.byte 4, 2, 8, 4, 6, 2, 8, 0, 1, 8, 3, 5, 1, 7, 0, 7, 0, 5, 2, 7, 8, 3, 1, 8, 3, 9, 4, 2, 5, 8, 8, 2, 1, 4, 5, 5, 2, 1, 2, 2, 7, 2, 5, 1, 2, 5, 0, 3, 2, 7
.byte 5, 5, 1, 2, 1, 6, 0, 3, 5, 4, 6, 9, 8, 1, 2, 0, 0, 5, 8, 1, 7, 6, 2, 1, 6, 5, 2, 1, 2, 8, 2, 7, 6, 5, 2, 7, 5, 1, 6, 9, 1, 2, 9, 6, 8, 9, 7, 7, 8, 9
.byte 3, 2, 2, 3, 8, 1, 9, 5, 7, 3, 4, 3, 2, 9, 3, 3, 9, 9, 4, 6, 4, 3, 7, 5, 0, 1, 9, 0, 7, 8, 3, 6, 9, 4, 5, 7, 6, 5, 8, 8, 3, 3, 5, 2, 3, 9, 9, 8, 8, 6
.byte 7, 5, 5, 0, 6, 1, 6, 4, 9, 6, 5, 1, 8, 4, 7, 7, 5, 1, 8, 0, 7, 3, 8, 1, 6, 8, 8, 3, 7, 8, 6, 1, 0, 9, 1, 5, 2, 7, 3, 5, 7, 9, 2, 9, 7, 0, 1, 3, 3, 7
.byte 6, 2, 1, 7, 7, 8, 4, 2, 7, 5, 2, 1, 9, 2, 6, 2, 3, 4, 0, 1, 9, 4, 2, 3, 9, 9, 6, 3, 9, 1, 6, 8, 0, 4, 4, 9, 8, 3, 9, 9, 3, 1, 7, 3, 3, 1, 2, 7, 3, 1
.byte 3, 2, 9, 2, 4, 1, 8, 5, 7, 0, 7, 1, 4, 7, 3, 4, 9, 5, 6, 6, 9, 1, 6, 6, 7, 4, 6, 8, 7, 6, 3, 4, 6, 6, 0, 9, 1, 5, 0, 3, 5, 9, 1, 4, 6, 7, 7, 5, 0, 4
.byte 9, 9, 5, 1, 8, 6, 7, 1, 4, 3, 0, 2, 3, 5, 2, 1, 9, 6, 2, 8, 8, 9, 4, 8, 9, 0, 1, 0, 2, 4, 2, 3, 3, 2, 5, 1, 1, 6, 9, 1, 3, 6, 1, 9, 6, 2, 6, 6, 2, 2
.byte 7, 3, 2, 6, 7, 4, 6, 0, 8, 0, 0, 5, 9, 1, 5, 4, 7, 4, 7, 1, 8, 3, 0, 7, 9, 8, 3, 9, 2, 8, 6, 8, 5, 3, 5, 2, 0, 6, 9, 4, 6, 9, 4, 4, 5, 4, 0, 7, 2, 4
.byte 7, 6, 8, 4, 1, 8, 2, 2, 5, 2, 4, 6, 7, 4, 4, 1, 7, 1, 6, 1, 5, 1, 4, 0, 3, 6, 4, 2, 7, 9, 8, 2, 2, 7, 3, 3, 4, 8, 0, 5, 5, 5, 5, 6, 2, 1, 4, 8, 1, 8
.byte 9, 7, 1, 4, 2, 6, 1, 7, 9, 1, 0, 3, 4, 2, 5, 9, 8, 6, 4, 7, 2, 0, 4, 5, 1, 6, 8, 9, 3, 9, 8, 9, 4, 2, 2, 1, 7, 9, 8, 2, 6, 0, 8, 8, 0, 7, 6, 8, 5, 2
.byte 8, 7, 7, 8, 3, 6, 4, 6, 1, 8, 2, 7, 9, 9, 3, 4, 6, 3, 1, 3, 7, 6, 7, 7, 5, 4, 3, 0, 7, 8, 0, 9, 3, 6, 3, 3, 3, 3, 0, 1, 8, 9, 8, 2, 6, 4, 2, 0, 9, 0
.byte 1, 0, 8, 4, 8, 8, 0, 2, 5, 2, 1, 6, 7, 4, 6, 7, 0, 8, 8, 3, 2, 1, 5, 1, 2, 0, 1, 8, 5, 8, 8, 3, 5, 4, 3, 2, 2, 3, 8, 1, 2, 8, 7, 6, 9, 5, 2, 7, 8, 6
.byte 7, 1, 3, 2, 9, 6, 1, 2, 4, 7, 4, 7, 8, 2, 4, 6, 4, 5, 3, 8, 6, 3, 6, 9, 9, 3, 0, 0, 9, 0, 4, 9, 3, 1, 0, 3, 6, 3, 6, 1, 9, 7, 6, 3, 8, 7, 8, 0, 3, 9
.byte 6, 2, 1, 8, 4, 0, 7, 3, 5, 7, 2, 3, 9, 9, 7, 9, 4, 2, 2, 3, 4, 0, 6, 2, 3, 5, 3, 9, 3, 8, 0, 8, 3, 3, 9, 6, 5, 1, 3, 2, 7, 4, 0, 8, 0, 1, 1, 1, 1, 6
.byte 6, 6, 6, 2, 7, 8, 9, 1, 9, 8, 1, 4, 8, 8, 0, 8, 7, 7, 9, 7, 9, 4, 1, 8, 7, 6, 8, 7, 6, 1, 4, 4, 2, 3, 0, 0, 3, 0, 9, 8, 4, 4, 9, 0, 8, 5, 1, 4, 1, 1
.byte 6, 0, 6, 6, 1, 8, 2, 6, 2, 9, 3, 6, 8, 2, 8, 3, 6, 7, 6, 4, 7, 4, 4, 7, 7, 9, 2, 3, 9, 1, 8, 0, 3, 3, 5, 1, 1, 0, 9, 8, 9, 0, 6, 9, 7, 9, 0, 7, 1, 4
.byte 8, 5, 7, 8, 6, 9, 4, 4, 0, 8, 9, 5, 5, 2, 9, 9, 0, 6, 5, 3, 6, 4, 0, 4, 4, 7, 4, 2, 5, 5, 7, 6, 0, 8, 3, 6, 5, 9, 9, 7, 6, 6, 4, 5, 7, 9, 5, 0, 9, 6
.byte 6, 6, 0, 2, 4, 3, 9, 6, 4, 0, 9, 9, 0, 5, 3, 8, 9, 6, 0, 7, 1, 2, 0, 1, 9, 8, 2, 1, 9, 9, 7, 6, 0, 4, 7, 5, 9, 9, 4, 9, 0, 1, 9, 7, 2, 3, 0, 2, 9, 7
.byte 6, 4, 9, 1, 3, 9, 8, 2, 6, 8, 0, 0, 3, 2, 9, 7, 3, 1, 5, 6, 0, 3, 7, 1, 2, 0, 0, 4, 1, 3, 7, 7, 9, 0, 3, 7, 8, 5, 5, 6, 6, 0, 8, 5, 0, 8, 9, 2, 5, 2
.byte 1, 6, 7, 3, 0, 9, 3, 9, 3, 1, 9, 8, 7, 2, 7, 5, 0, 2, 7, 5, 4, 6, 8, 9, 0, 6, 9, 0, 3, 7, 0, 7, 5, 3, 9, 4, 1, 3, 0, 4, 2, 6, 5, 2, 3, 1, 5, 0, 1, 1
.byte 9, 4, 8, 0, 9, 3, 7, 7, 2, 4, 5, 0, 4, 8, 7, 9, 5, 1, 5, 0, 9, 5, 4, 1, 0, 0, 9, 2, 1, 6, 4, 5, 8, 6, 3, 7, 5, 4, 7, 1, 0, 5, 9, 8, 4, 3, 6, 7, 9, 1
.byte 7, 8, 6, 3, 9, 1, 6, 7, 0, 2, 1, 1, 8, 7, 4, 9, 2, 4, 3, 1, 9, 9, 5, 7, 0, 0, 6, 4, 1, 9, 1, 7, 9, 6, 9, 7, 7, 7, 5, 9, 9, 0, 2, 8, 3, 0, 0, 6, 9, 9
.byte 1, 5, 3, 6, 8, 7, 1, 3, 7, 1, 1, 9, 3, 6, 6, 1, 4, 9, 5, 2, 8, 1, 1, 3, 0, 5, 8, 7, 6, 3, 8, 0, 2, 7, 8, 4, 1, 0, 7, 5, 4, 4, 4, 9, 7, 3, 3, 0, 7, 8
.byte 4, 0, 7, 8, 9, 9, 2, 3, 1, 1, 5, 5, 3, 5, 5, 6, 2, 5, 6, 1, 1, 4, 2, 3, 2, 2, 4, 2, 3, 2, 5, 5, 0, 3, 3, 6, 8, 5, 4, 4, 2, 4, 8, 8, 9, 1, 7, 3, 5, 3
.byte 4, 4, 8, 8, 9, 9, 1, 1, 5, 0, 1, 4, 4, 0, 6, 4, 8, 0, 2, 0, 3, 6, 9, 0, 6, 8, 0, 6, 3, 9, 6, 0, 6, 7, 2, 3, 2, 2, 1, 9, 3, 2, 0, 4, 1, 4, 9, 5, 3, 5
.byte 4, 1, 5, 0, 3, 1, 2, 8, 8, 8, 0, 3, 3, 9, 5, 3, 6, 0, 5, 3, 2, 9, 9, 3, 4, 0, 3, 6, 8, 0, 0, 6, 9, 7, 7, 7, 1, 0, 6, 5, 0, 5, 6, 6, 6, 3, 1, 9, 5, 4
.byte 8, 1, 2, 3, 4, 8, 8, 0, 6, 7, 3, 2, 1, 0, 1, 4, 6, 7, 3, 9, 0, 5, 8, 5, 6, 8, 5, 5, 7, 9, 3, 4, 5, 8, 1, 4, 0, 3, 6, 2, 7, 8, 2, 2, 7, 0, 3, 2, 8, 0
.byte 8, 2, 6, 1, 6, 5, 7, 0, 7, 7, 3, 9, 4, 8, 3, 2, 7, 5, 9, 2, 2, 3, 2, 8, 4, 5, 9, 4, 1, 7, 0, 6, 5, 2, 5, 0, 9, 4, 5, 1, 2, 3, 2, 5, 2, 3, 0, 6, 0, 8
.byte 2, 2, 9, 1, 8, 8, 0, 2, 0, 5, 8, 7, 7, 7, 3, 1, 9, 7, 1, 9, 8, 3, 9, 4, 5, 0, 1, 8, 0, 8, 8, 8, 0, 7, 2, 4, 2, 9, 6, 6, 1, 9, 8, 0, 8, 1, 1, 1, 9, 7
.byte 7, 7, 1, 5, 8, 5, 4, 2, 5, 0, 2, 0, 1, 6, 5, 4, 5, 0, 9, 0, 4, 1, 3, 2, 4, 5, 8, 0, 9, 7, 8, 6, 8, 8, 2, 7, 7, 8, 9, 4, 8, 7, 2, 1, 8, 5, 9, 6, 1, 7
.byte 7, 2, 1, 0, 7, 8, 3, 8, 4, 3, 5, 0, 6, 9, 1, 8, 6, 1, 5, 5, 4, 3, 5, 6, 6, 2, 8, 8, 4, 0, 6, 2, 2, 5, 7, 4, 7, 3, 6, 9, 2, 2, 8, 4, 5, 0, 9, 5, 1, 6
.byte 2, 0, 8, 4, 9, 6, 0, 3, 9, 8, 0, 1, 3, 4, 0, 0, 1, 7, 2, 3, 9, 3, 0, 6, 7, 1, 6, 6, 6, 8, 2, 3, 5, 5, 5, 2, 4, 5, 2, 5, 2, 8, 0, 4, 6, 0, 9, 7, 2, 2
.byte 5, 3, 5, 0, 3, 5, 3, 4, 2, 2, 6, 4, 7, 2, 5, 2, 4, 2, 5, 0, 8, 7, 4, 0, 5, 4, 0, 7, 5, 5, 9, 1, 7, 8, 9, 7, 8, 1, 2, 6, 4, 3, 3, 0, 3, 3, 1, 6, 9, 0

.section .rodata
        .align  2
numstring:
        .asciz "%d"
nlstring:
        .asciz "\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
```

```
        stmfd   sp!, {r4-r10, lr}

        ldr     col_num, =col_nums
loopstart:
        ldr     row1, =buffer
        ldr     tmp, =col_nums
        add     row1, row1, tmp
col_loop:
        mov     col_sum, 0
        ldr     row_num, =row_nums
        sub     row1, row1, 1
        mov     data_ptr, row1
        subs    col_num, col_num, 1
        blt     printme

row_loop:
        ldrb    byte, [data_ptr], col_offset
        add     col_sum, col_sum, byte
        subs    row_num, row_num, 1
        bne     row_loop

        mov     r0, col_num
        bl      get_3_result

        mov     tmp, r0
        add     col_sum, col_sum, tmp
        mov     r0, col_sum
        mov     r1, col_num
        bl      put_3_result

        b       col_loop
printme:
        ldr     row1, =ten
        ldr     data_ptr, =result

next_digit:
        ldrb    r1, [data_ptr], 1
        ldr     r0, =numstring
        bl      printf
        subs    row1, row1, 1
        bne     next_digit
last:
        ldr     r0, =nlstring
        bl      printf
        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

        .global get_3_result
        .type   get_3_result, %function
get_3_result:
        stmfd   sp!, {data_ptr, tmp, lr}
        ldr     data_ptr, =result
        add     data_ptr, data_ptr, r0
        ldrb    byte, [data_ptr], 1
        mov     const, hundred
        mul     byte, byte, const
        mov     tmp, byte
        ldrb    byte, [data_ptr], 1
        mov     const, ten
        mul     byte, byte, const
        add     tmp, tmp, byte
        ldrb    byte, [data_ptr], 1
        add     tmp, tmp, byte
        mov     r0, tmp
        ldmfd   sp!, {data_ptr, tmp, pc}

        .global put_3_result
        .type   put_3_result, %function
put_3_result:
        stmfd   sp!, {data_ptr, lr}
        ldr     data_ptr, =result
        add     data_ptr, data_ptr, r1
        add     data_ptr, data_ptr, 2

        bl      divide_by_10_remainder

        strb    r1, [data_ptr], -1
        bl      divide_by_10_remainder
        strb    r1, [data_ptr], -1
        strb    r0, [data_ptr]
        ldmfd   sp!, {data_ptr, pc}
```

divide_by_10.s

```
.syntax unified

# this subroutine divides the passed number by 10 and
# returns the dividend and remainder
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xcccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
# and subtract from the original number to give the remainder on division
# by 10.
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend
#   r1 - the remainder

.equ const,-0x33333333
#.equ const,0xcccccccd
.text
.align  2
.global divide_by_10_remainder
.type   divide_by_10_remainder, %function
divide_by_10_remainder:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     rsmall
```

```
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r3, r2          @ r3 = r2
        mov     r3, r3, asl #2  @ r3 = 4 * r3
        add     r3, r3, r2      @ r3 = r3 + r2
        mov     r3, r3, asl #1  @ r3 = 2 * r3
        rsb     r3, r3, r0      @ r3 = r0 - r3 = r0 - 10*int(r0/10)
        mov     r1, r3          @ the remainder
        mov     r0, r2          @ the dividend
        b       rlast
rsmall:
        mov     r1, r0
        mov     r0, 0
rlast:
        ldmfd   sp!, {pc}


# this subroutine divides the passed number by 10
# returns the dividend
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xcccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend

.align  2
.global divide_by_10
.type   divide_by_10, %function
divide_by_10:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     small
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r0, r2          @ the dividend
        b       last
small:
        mov     r0, 0
last:
        ldmfd   sp!, {pc}
```

Description of problem

The following iterative sequence is defined for the set of positive integers:

n â†’ *n*/2 (*n* is even)
n â†’ 3*n* + 1 (*n* is odd)

Using the rule above and starting with 13, we generate the following sequence:

13 â†’ 40 â†’ 20 â†’ 10 â†’ 5 â†’ 16 â†’ 8 â†’ 4 â†’ 2 â†’ 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Which starting number, under one million, produces the longest chain?

NOTE: Once the chain starts the terms are allowed to go above one million.

pe014.s

```
.syntax unified

.equ last, 1000000
.equ first, 500000

.align 4

maxi            .req r4
maxv            .req r5
counter         .req r6
i               .req r7
j_hi            .req r8
j_lo            .req r9
limit           .req r10

.section .rodata
        .align  2
numstring:
        .asciz "%d\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r10, lr}

        mov     maxi, 0
        mov     maxv, 0
        ldr     i, =first
        ldr     limit, =last
loopstart:
        mov     counter, 0
        mov     j_lo, i
inner:
        cmp     j_lo, 1
        beq     inc_counter
        mov     r0, j_hi
        mov     r1, j_lo
        bl      next_term
        mov     j_hi, r0
        mov     j_lo, r1
        add     counter, counter, 1
        b       inner
```

```
inc_counter:
        add     counter, counter, 1
        cmp     maxv, counter
        movlt   maxv, counter
        movlt   maxi, i
        adds    i, i, 1
        cmp     i, limit
        blt     loopstart

printme:
        mov     r1, maxi
        ldr     r0, =numstring
        bl      printf
        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

        .global next_term
        .type   next_term, %function
next_term:
        stmfd   sp!, {lr}
        ands    r2, r1, 1
        bne     odd
        movs    r0, r0, lsr 1   @ leave bit 0 in carry
        rrx     r1, r1          @ then get it and push it on
        b       next_term_last
odd:
        adds    r2, r1, r1
        adc     r0, r0, 0
        adds    r2, r2, r1
        adc     r0, r0, 0
        adds    r2, r2, 1
        adc     r0, r0, 0
        mov     r1, r2          @ r1 is 3n+1 - any overflow to r0

next_term_last:
        ldmfd   sp!, {pc}
```

Description of problem

Starting in the top left corner of a 2Ã—2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.

How many such routes are there through a 20Ã—20 grid?

pe015.s

```
.syntax unified
.align 4

.macro factor2 d n
        ldr     i, =start_offset
inext\@:
        ldr     i_ptr, =\d
        add     i_ptr, i_ptr, i
        ldrb    dtmp, [i_ptr], -1
        ldr     j, =start_offset
jnext\@:
        cmp     dtmp, 1
        beq     nexti\@
        ldr     j_ptr, =\n
        add     j_ptr, j_ptr, j
        ldrb    ntmp, [j_ptr], -1

        cmp     ntmp, dtmp      @ if numerator < denominator use next numerator element
        blt     nextj\@
        mov     tmp, 0
        mov     r0, 0
mod_start\@:
        add     r0, r0, 1
        add     tmp, tmp, dtmp
        cmp     ntmp, tmp
        bgt     mod_start\@
        blt     nextj\@
        strb    r0, [j_ptr, 1]
        mov     dtmp, 1
        strb    dtmp, [i_ptr, 1]
nextj\@:
        subs    j, j, 1
        bgt     jnext\@
nexti\@:
        subs    i, i, 1
        bgt     inext\@
.endm

.equ num, 20
.equ start_offset, 19

i               .req r4
j               .req r5
i_ptr           .req r6
j_ptr           .req r7
res_hi          .req r6
res_lo          .req r7
tmp             .req r8
dtmp            .req r9
ntmp            .req r10

.section .data
numerator:
  .byte 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
denominator:
  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
.section .rodata
        .align  2
llustring:
        .asciz "%llu\n"
.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r10, lr}
```

```
        factor2 denominator numerator
nloop:
        ldr     r0, =numerator
        bl      needs_factor
        cmp     r0, 0
        beq     printme
        factor2 numerator denominator
        ldr     r0, =denominator
        bl      needs_factor
        cmp     r0, 0
        beq     printme
        factor2 denominator numerator
        b       nloop

printme:
        ldr     j, =start_offset
        ldr     j_ptr, =numerator
        add     j_ptr, j_ptr, j
        mov     r0, 1
        mov     r1, 0
mnumerator:
        ldrb    ntmp, [j_ptr], -1
        mov     r2, ntmp
        mov     r3, 0
        bl      mul_64to64
        subs    j, j, 1
        bge     mnumerator

        mov     r2, r0
        mov     r3, r1
        ldr     r0, =llustring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

        .align  2
        .global needs_factor
        .type   needs_factor, %function
needs_factor:
        stmfd   sp!, {lr}
        ldr     r1, =start_offset
        add     r2, r0, r1
next_byte:
        ldrb    r3, [r2], -1
        cmp     r3, 1
        bne     ret1
        subs    r1, r1, 1
        bne     next_byte
        mov     r0, 0
        b       leave
ret1:
        mov     r0, 1
leave:
        ldmfd   sp!, {pc}
```

mul_64to64.s

```
# long long mul_64to64 (long long b, long long c)

b_0 .req r0     @ b bits [31:00]  (b low)
b_1 .req r1     @ b bits [63:32]  (b high)
c_0 .req r2     @ c bits [31:00]  (c low)
c_1 .req r3     @ c bits [63:32]  (c high)
a_0 .req r4     @ a bits [31:00]  (a low-low)
a_1 .req r5     @ a bits [63:32]  (a low-high)

.globl mul_64to64
.align 2
        .text
mul_64to64:

        nop
        stmfd sp!, {r4, r5, lr}         @ 64-bit a = 64-bit b * 64-bit c

        umull a_0, a_1, b_0, c_0        @ low*low
        mla a_1, b_0, c_1, a_1          @ low*high
        mla a_1, b_1, c_0, a_1          @ high*low
                                        @ return wrapper
        mov r0, a_0
        mov r1, a_1

        ldmfd sp!, {r4, r5, pc}
```

Description of problem

$2^{15} = 32768$ and the sum of its digits is 3 + 2 + 7 + 6 + 8 = 26.

What is the sum of the digits of the number 2^{1000}?

pe016.s

```
.syntax unified

.equ    power,1000
.equ    iLENGTH,302
.equ    scalar2,2

.section .rodata
sum_string:
.asciz "%d\n"

.section .data
input:
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

.section bss
.lcomm output,iLENGTH

.text
```

```
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4, lr}
        ldr     r4, =power
next:
        ldr     r0, =input
        ldr     r1, =iLENGTH
        ldr     r2, =scalar2
        ldr     r3, =output
        bl      mul_digit_string

        ldr     r0, =output
        ldr     r1, =iLENGTH
        ldr     r2, =input
        bl      copybytes

        subs    r4, r4, 1
        bne     next

        ldr     r0, =output
        ldr     r1, =iLENGTH
        bl      sum_output

        mov     r1, r0
        ldr     r0, =sum_string
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

# printbytes takes input pointer in r0, input length in r1 and writes printable vector to r2 (with trailing null)
printbytes:
        stmfd   sp!, {lr}
printloop:
        ldrb    r3, [r0], 1
        add     r3, r3, 48
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     printloop

        mov     r3, 0
        strb    r3, [r2], 1
        ldmfd   sp!, {pc}

# copybytes takes input pointer in r0, input length in r1 and copies vector to r2
copybytes:
        stmfd   sp!, {lr}
copyloop:
        ldrb    r3, [r0], 1
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     copyloop

        ldmfd   sp!, {pc}

# sum_output sums the elements of the r1 elements of the vector passed in r0 and returns the sum in r0
sum_output:
        stmfd   sp!, {lr}
        mov     r2, 0
sumloop:
        ldrb    r3, [r0], 1
        add     r2, r2, r3
        subs    r1, r1, 1
        bne     sumloop

        mov     r0, r2
        ldmfd   sp!, {pc}
```

mul_digit_string.s

```
.syntax unified

# this subroutine multiplies the byte array at r0, length r1 by the digit r2
# and stores to r0 with output length in r1
#
# inputs
#   r0 - pointer to input vector
#   r1 - length of input vector
#   r2 - multiplicand
#   r3 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

iptr            .req r4
optr            .req r5
offset          .req r6
tmp             .req r7
carry           .req r8
multiplier      .req r9
cell            .req r10

.global mul_digit_string
.type mul_digit_string, %function
.text
.align  2

mul_digit_string:
        stmfd   sp!, {r4-r10, lr}
        teq     r2, 0
        bne     mds_one
        moveq   r0, r3
        moveq   tmp, r3
        moveq   r1, 1
        bleq    clearbytes
        mov     r0, tmp
        moveq   r1, 1
        b       mds_end
mds_one:
        teq     r2, 1
        bne     mds_start
        moveq   r2, r3
```

```
        moveq   tmp, r0
        moveq   cell, r1
        bleq    copybytes
        mov     r0, tmp
        mov     r1, cell
        b       mds_end
mds_start:
        mov     carry, r0
        mov     tmp, r1
        mov     offset, r3
        mov     r0, r3
        add     r1, r1, 1
        bl      clearbytes
        mov     r0, carry
        mov     r1, tmp
        mov     r3, offset

        mov     carry, 0
        mov     multiplier, r2
        mov     offset, r1
        sub     offset, offset, 1
        add     iptr, r0, offset
        add     optr, r3, offset
        add     offset, offset, 1
        mov     tmp, r1
mds_loopstart:
        ldrb    cell, [iptr], -1
        mul     r0, cell, multiplier
        add     r0, r0, carry
        bl      divide_by_10_remainder
        strb    r1, [optr], -1
        mov     carry, r0
        subs    offset, offset, 1
        beq     mds_last
        b       mds_loopstart
mds_last:
        cmp     carry, 0
        addeq   r0, optr, 1
        moveq   r1, tmp
        strbne  carry, [optr]
        movne   r0, optr
        addne   r1, tmp, 1
mds_end:
        ldmfd   sp!, {r4-r10, pc}
```

divide_by_10.s

```
.syntax unified

# this subroutine divides the passed number by 10 and
# returns the dividend and remainder
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xcccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
# and subtract from the original number to give the remainder on division
# by 10.
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend
#   r1 - the remainder

.equ const,-0x33333333
#.equ const,0xcccccccd
.text
.align  2
.global divide_by_10_remainder
.type   divide_by_10_remainder, %function
divide_by_10_remainder:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     rsmall
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r3, r2          @ r3 = r2
        mov     r3, r3, asl #2  @ r3 = 4 * r3
        add     r3, r3, r2      @ r3 = r3 + r2
        mov     r3, r3, asl #1  @ r3 = 2 * r3
        rsb     r3, r3, r0      @ r3 = r0 - r3 = r0 - 10*int(r0/10)
        mov     r1, r3          @ the remainder
        mov     r0, r2          @ the dividend
        b       rlast
rsmall:
        mov     r1, r0
        mov     r0, 0
rlast:
        ldmfd   sp!, {pc}


# this subroutine divides the passed number by 10
# returns the dividend
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xcccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend

.align  2
.global divide_by_10
.type   divide_by_10, %function
divide_by_10:
        stmfd   sp!, {lr}
```

```
        cmp     r0, 10
        blt     small
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r0, r2          @ the dividend
        b       last
small:
        mov     r0, 0
last:
        ldmfd   sp!, {pc}
```

copybytes.s

```
.syntax unified

# copybytes takes input pointer in r0, input length in r1 and writes to r2
ptr     .req r4
len     .req r5
.global copybytes
.type   copybytes, %function
copybytes:
        stmfd   sp!, {ptr, len, lr}
        mov     len, r1
        mov     ptr, r2
copybytesloopstart:
        ldrb    r3, [r0], 1
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     copybytesloopstart

        mov     r0, ptr
        mov     r1, len
        ldmfd   sp!, {ptr, len, pc}
```

clearbytes.s

```
.syntax unified

# clearbytes takes pointer in r0, input length in r1
# and sets all vector elements to 0

.global clearbytes
.type   clearbytes, %function
clearbytes:
        stmfd   sp!, {lr}
        mov     r3, 0
clearbytesloopstart:
        strb    r3, [r0], 1
        subs    r1, r1, 1
        bne     clearbytesloopstart

        ldmfd   sp!, {pc}
```

Description of problem

If the numbers 1 to 5 are written out in words: one, two, three, four, five, then there are 3 + 3 + 5 + 4 + 4 = 19 letters used in total.

If all the numbers from 1 to 1000 (one thousand) inclusive were written out in words, how many letters would be used?

NOTE: Do not count spaces or hyphens. For example, 342 (three hundred and forty-two) contains 23 letters and 115 (one hundred and fifteen) contains 20 letters. The use of "and" when writing out numbers is in compliance with British usage.

pe017.s

```
.syntax unified

.equ    one,3
.equ    two,3
.equ    three,5
.equ    four,4
.equ    five,4
.equ    six,3
.equ    seven,5
.equ    eight,5
.equ    nine,4
.equ    ten,3
.equ    eleven,6
.equ    twelve,6
.equ    thirteen,8
.equ    fourteen,8
.equ    fifteen,7
.equ    sixteen,7
.equ    seventeen,9
.equ    eighteen,8
.equ    nineteen,8
.equ    twenty,6
.equ    thirty,6
.equ    forty,5
.equ    fifty,5
.equ    sixty,5
.equ    seventy,7
.equ    eighty,6
.equ    ninety,6
.equ    hundred,7
.equ    thousand,8
.equ    and,3

.macro units
        add     r0, r0, one
        add     r0, r0, two
        add     r0, r0, three
        add     r0, r0, four
        add     r0, r0, five
        add     r0, r0, six
        add     r0, r0, seven
        add     r0, r0, eight
        add     r0, r0, nine
```

```
.endm

.macro teens
        add     r0, r0, ten
        add     r0, r0, eleven
        add     r0, r0, twelve
        add     r0, r0, thirteen
        add     r0, r0, fourteen
        add     r0, r0, fifteen
        add     r0, r0, sixteen
        add     r0, r0, seventeen
        add     r0, r0, eighteen
        add     r0, r0, nineteen
.endm

.macro tens tenmul
        ldr     r1, =\tenmul
        mov     r2, 10
        mul     r1, r1, r2
        add     r0, r0, r1
        units
.endm

.macro alltens
        teens
        tens    twenty
        tens    thirty
        tens    forty
        tens    fifty
        tens    sixty
        tens    seventy
        tens    eighty
        tens    ninety
.endm

.macro hundreds hundredmul
        ldr     r1, =\hundredmul
        add     r1, r1, hundred
        mov     r2, 100
        mul     r1, r1, r2
        add     r0, r0, r1

        ldr     r1, =and
        mov     r2, 99
        mul     r1, r1, r2
        add     r0, r0, r1

        units
        alltens
.endm

.section .rodata
sum_string:
.asciz "%d\n"

.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4, lr}
        mov     r0, 0
        units
        alltens
        hundreds        one
        hundreds        two
        hundreds        three
        hundreds        four
        hundreds        five
        hundreds        six
        hundreds        seven
        hundreds        eight
        hundreds        nine
        add     r0, r0, one
        add     r0, r0, thousand

        mov     r1, r0
        ldr     r0, =sum_string
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3
7 4
2 **4** 6
8 5 **9** 3

That is, 3 + 7 + 4 + 9 = 23.

Find the maximum total from top to bottom of the triangle below:

75
95 64
17 47 82
18 35 87 10
20 04 82 47 65
19 01 23 75 03 34
88 02 77 73 07 63 67
99 65 04 28 06 16 70 92
41 41 26 56 83 40 80 70 33
41 48 72 33 47 32 37 16 94 29
53 71 44 65 25 43 91 52 97 51 14
70 11 33 28 77 73 17 78 39 68 17 57
91 71 52 38 17 14 91 43 58 50 27 29 48

63 66 04 68 89 53 67 30 73 16 69 87 40 31
04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

NOTE: As there are only 16384 routes, it is possible to solve this problem by trying every route. However, Problem 67, is the same challenge with a triangle containing one-hundred rows; it cannot be solved by brute force, and requires a clever method! ;o)

```
pe018.s

.syntax unified

.equ    maxij, 14
.equ    width, 2
.equ    logwidth, 1

.align 4

iptr            .req r0
tmp             .req r1
icount          .req r4
jcount          .req r5
maxc            .req r6
jptr            .req r7
cell            .req r8

.macro get_element i, j
        ldr     iptr, =last
        mov     r1, \i
        add     iptr, iptr, r1
        sub     iptr, iptr, 1
        ldrb    tmp, [iptr]

        ldr     jptr, =buffer
        add     jptr, jptr, tmp, asl logwidth
        mov     tmp, \j
        add     jptr, jptr, tmp, asl logwidth
        sub     jptr, jptr, width
        ldrh    cell, [jptr]
.endm

.macro update_element i, j
        mov     r2, \i
        add     r2, r2, 1
        mov     r3, \j
        add     r3, r3, 1
        get_element r2 r3
        sub     r3, r3, 1
        mov     maxc, cell
        get_element r2 r3
        cmp     maxc, cell
        movlt   maxc, cell
        sub     r2, r2, 1
        get_element r2 r3
        add     cell, cell, maxc
        strh    cell, [jptr]
.endm

.section .data
last:
.byte 0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120

buffer:
.hword 75
.hword 95, 64
.hword 17, 47, 82
.hword 18, 35, 87, 10
.hword 20, 4, 82, 47, 65
.hword 19, 1, 23, 75, 3, 34
.hword 88, 2, 77, 73, 7, 63, 67
.hword 99, 65, 4, 28, 6, 16, 70, 92
.hword 41, 41, 26, 56, 83, 40, 80, 70, 33
.hword 41, 48, 72, 33, 47, 32, 37, 16, 94, 29
.hword 53, 71, 44, 65, 25, 43, 91, 52, 97, 51, 14
.hword 70, 11, 33, 28, 77, 73, 17, 78, 39, 68, 17, 57
.hword 91, 71, 52, 38, 17, 14, 91, 43, 58, 50, 27, 29, 48
.hword 63, 66, 4, 68, 89, 53, 67, 30, 73, 16, 69, 87, 40, 31
.hword 4, 62, 98, 27, 23, 9, 70, 98, 73, 93, 38, 53, 60, 4, 23

.section .rodata
.align  2
resstring:
        .asciz "%d\n"

.text
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4-r8, lr}

        ldr     icount, =maxij
iloop:
        mov     jcount, icount
jloop:
        update_element icount jcount
        subs    jcount, jcount, 1
        bne     jloop
        subs    icount, icount, 1
        bne     iloop
printme:
        mov     r1, cell
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r8, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

Description of problem

You are given the following information, but you may prefer to do some research for yourself.

- 1 Jan 1900 was a Monday.

- Thirty days has September,
  April, June and November.
  All the rest have thirty-one,
  Saving February alone,
  Which has twenty-eight, rain or shine.
  And on leap years, twenty-nine.
- A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400.

How many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

pe019.s

```
.syntax unified

.equ    months,48
.equ    cycles,25

tmp             .req r1
scount          .req r4
ccount          .req r5
mcount          .req r6
dow             .req r7
cptr            .req r8

.section .rodata
cycle:
.byte 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

resstring:
.asciz "%d\n"

.text
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4-r8, lr}
        mov     scount, 0
        mov     dow, 2
        ldr     ccount, =cycles
cstart:
        mov     mcount, 0
        ldr     cptr, =cycle
mstart:
        teq     dow, 0
        addeq   scount, scount, 1

        ldrb    tmp, [cptr], 1
        add     r0, dow, tmp
        mov     r1, 7
        bl      divide
        mov     dow, r1

        add     mcount, mcount, 1
        cmp     mcount, months
        blt     mstart
        subs    ccount, ccount, 1
        bne     cstart
last:
        mov     r1, scount
        ldr     r0, =resstring
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r8, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

n! means *n* Ã— (*n* âˆ' 1) Ã— ... Ã— 3 Ã— 2 Ã— 1

For example, 10! = 10 Ã— 9 Ã— ... Ã— 3 Ã— 2 Ã— 1 = 3628800,
and the sum of the digits in the number 10! is 3 + 6 + 2 + 8 + 8 + 0 + 0 = 27.

Find the sum of the digits in the number 100!

pe020.s

```
.syntax unified

.equ    LENGTH,200
.equ    scalar100,100

.section .rodata
sumstring:
.asciz "%d\n"


.section bss
.align  2
.lcomm input,LENGTH
.lcomm output,LENGTH

.text
```

```
        .align  2
        .global main
        .type   main, %function
main:
        stmfd   sp!, {r4, lr}
        mov     r3, 1
        ldr     r0, =input
        strb    r3, [r0]

        ldr     r0, =scalar100
        ldr     r1, =input
        ldr     r2, =output
        bl      factorial

        mov     r2, 0
lstart:
        ldrb    r3, [r0], 1
        add     r2, r2, r3
        subs    r1, r1, 1
        bne     lstart

        mov     r1, r2
        ldr     r0, =sumstring
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

factorial.s

```
.syntax unified

# this computes the factorial of the number passed
#
# inputs
#   r0 - the number used for factorial
#   r1 - pointer to input vector of bytes
#   r2 - pointer to output vector of bytes
#
# outputs
#   r0 - output vector pointer
#   r1 - length of output vector

number  .req r4
iptr    .req r5
ilen    .req r6
tmp     .req r7
counter .req r8
optr    .req r9

.text
        .align  2
        .global factorial
        .type   factorial, %function
factorial:
        stmfd   sp!, {r4-r9, lr}

        cmp     r0, 2
        bge     factorial_ok
        mov     r0, r1
        mov     r1, 1
        bl      copybytes
        b       factorial_end
factorial_ok:
        mov     number, r0
        mov     iptr, r1
        mov     optr, r2
        mov     ilen, 1
        mov     counter, 2
        mov     r0, optr
factorial_start:
        mov     r0, optr
        mov     r1, ilen
        add     r1, r1, 1
bclear:
        bl      clearbytes
aclear:
        mov     r0, iptr
        mov     r1, ilen
        mov     r2, counter
        mov     r3, optr
bmis:
        bl      mul_int_string
amis:
        teq     counter, number
        beq     factorial_last
        mov     ilen, r1
        mov     r2, iptr
bcopy:
        bl      copybytes
acopy:
        add     counter, counter, 1
        b       factorial_start
factorial_last:
        mov     counter, r0
        mov     tmp, r1
        mov     r0, iptr
        mov     r1, ilen
        bl      clearbytes
        mov     r1, tmp
        mov     r0, counter
factorial_end:
        ldmfd   sp!, {r4-r9, pc}
```

mul_int_string.s

```
.syntax unified

.equ datum_size, 1
.equ MAXLEN,192

.section bss
```

```
.lcomm tmp_vector,MAXLEN

# this subroutine multiplies the byte array at r0, length r1 by the int r2
# and stores to r0 with output length in r1
#
# inputs
#   r0 - pointer to input vector
#   r1 - length of input vector
#   r2 - multiplicand
#   r3 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

iptr            .req r4
optr            .req r5
ilength         .req r6
tlength         .req r7
olength         .req r8
tmp             .req r9
multiplier      .req r10

.global mul_int_string
.type mul_int_string, %function
.text
.align  2

mul_int_string:
        stmfd   sp!, {r4-r10, lr}
        mov     iptr, r0
        mov     ilength, r1
        mov     tlength, r1
        mov     olength, r1
        mov     multiplier, r2
        mov     optr, r3
mis_loopstart:
        teq     multiplier, 0
        beq     mis_last

        ldr     r0, =tmp_vector
        mov     r1, tlength
        bl      clearbytes

        mov     r0, multiplier
        bl      divide_by_10_remainder
        mov     multiplier, r0
        mov     r2, r1
        mov     r0, iptr
        mov     r1, ilength
        ldr     r3, =tmp_vector
bm:
        bl      mul_digit_string
am:
        mov     tmp, r0
        cmp     tlength, r1
        movlt   tlength, r1    @ set tlength to max of tlength and r1

        stmfd   sp!, {r4}
        mov     r0, optr
        stmfd   sp!, {r0}       @ this is the fifth parameter for the subroutine

        mov     r0, tmp
        mov     r1, tlength
        mov     r2, optr
ba:
        mov     r3, olength
        bl      add_digit_strings
        mov     optr, r0
        cmp     olength, r1
        movlt   olength, r1
        add     sp, sp, 4       @ revert sp to before (1)
        ldmfd   sp!, {r4}
aa:
        ldr     r0, =tmp_vector
        mov     r1, tlength
        bl      clearbytes
        mov     r0, optr
        mov     r1, olength
        teq     multiplier, 0
        addne   ilength, ilength, 1
        beq     mis_last
        bne     mis_loopstart
mis_last:
        ldmfd   sp!, {r4-r10, pc}
```

add_digit_strings.s

```
.syntax unified

# see usage in test_add_digit_strings.s - it requires
# the fifth parameter to be passed on the stack

.equ datum_size, 1

.text
optr            .req r4
sptr            .req r5
lptr            .req r6
scell           .req r7
lcell           .req r8
carry           .req r9
ltmp            .req r9
counter         .req r10
ptmp            .req r10

# this subroutine adds the byte array at r0, length r1
# to the byte array at r2, length r3. The data is output
# to the pointer passed as r4.
#
# inputs
#   r0 - pointer to input1 vector
#   r1 - length of input1 vector
#   r2 - pointer to input2 vector
#   r3 - length of input2 vector
#   r4 - pointer to output vector
#
```

```
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

.text
.align  2

.global add_digit_strings
.type add_digit_strings, %function
add_digit_strings:
        stmfd   sp!, {r9-r10, lr}
        cmp     r3, r1
        movlt   ptmp, r0
        movlt   r0, r2
        movlt   r2, ptmp
        movlt   ltmp, r1
        movlt   r1, r3
        movlt   r3, ltmp
        bl      add_strings_short_to_long
        ldmfd   sp!, {r9-r10, pc}

# this subroutine adds the short byte array at r0, length r1
# to the byte array at r2, length r3.
#
# inputs
#   r0 - pointer to short vector
#   r1 - length of short vector
#   r2 - pointer to long vector
#   r3 - length of long vector
#   r4 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

.global add_strings_short_to_long
.type add_strings_short_to_long, %function
add_strings_short_to_long:
        stmfd   sp!, {r5-r10, lr} @ 7 longs
        ldr     optr, [sp, #40]
        mov     sptr, r0
        add     sptr, sptr, r1
        sub     sptr, sptr, 1

        mov     lptr, r2
        add     lptr, lptr, r3
        sub     lptr, lptr, 1

        add     optr, optr, r3

        mov     carry, 0
        mov     counter, r1
sstart:
        ldrb    scell, [sptr], -1
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, scell
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     sstart

        mov     counter, r3
        subs    counter, counter, r1
        beq     asstl_last
lstart:
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     lstart

asstl_last:
        cmp     carry, 1
        strbeq  carry, [optr], -1

        add     r0, optr, 1
        add     r1, r3, carry
        ldmfd   sp!, {r5-r10, pc}
```

mul_digit_string.s

```
.syntax unified

# this subroutine multiplies the byte array at r0, length r1 by the digit r2
# and stores to r0 with output length in r1
#
# inputs
#   r0 - pointer to input vector
#   r1 - length of input vector
#   r2 - multiplicand
#   r3 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

iptr            .req r4
optr            .req r5
offset          .req r6
tmp             .req r7
carry           .req r8
multiplier      .req r9
cell            .req r10

.global mul_digit_string
.type mul_digit_string, %function
.text
.align  2
```

```
mul_digit_string:
        stmfd   sp!, {r4-r10, lr}
        teq     r2, 0
        bne     mds_one
        moveq   r0, r3
        moveq   tmp, r3
        moveq   r1, 1
        bleq    clearbytes
        mov     r0, tmp
        moveq   r1, 1
        b       mds_end
mds_one:
        teq     r2, 1
        bne     mds_start
        moveq   r2, r3
        moveq   tmp, r0
        moveq   cell, r1
        bleq    copybytes
        mov     r0, tmp
        mov     r1, cell
        b       mds_end
mds_start:
        mov     carry, r0
        mov     tmp, r1
        mov     offset, r3
        mov     r0, r3
        add     r1, r1, 1
        bl      clearbytes
        mov     r0, carry
        mov     r1, tmp
        mov     r3, offset

        mov     carry, 0
        mov     multiplier, r2
        mov     offset, r1
        sub     offset, offset, 1
        add     iptr, r0, offset
        add     optr, r3, offset
        add     offset, offset, 1
        mov     tmp, r1
mds_loopstart:
        ldrb    cell, [iptr], -1
        mul     r0, cell, multiplier
        add     r0, r0, carry
        bl      divide_by_10_remainder
        strb    r1, [optr], -1
        mov     carry, r0
        subs    offset, offset, 1
        beq     mds_last
        b       mds_loopstart
mds_last:
        cmp     carry, 0
        addeq   r0, optr, 1
        moveq   r1, tmp
        strbne  carry, [optr]
        movne   r0, optr
        addne   r1, tmp, 1
mds_end:
        ldmfd   sp!, {r4-r10, pc}
```

divide_by_10.s

```
.syntax unified

# this subroutine divides the passed number by 10 and
# returns the dividend and remainder
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xccccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
# and subtract from the original number to give the remainder on division
# by 10.
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend
#   r1 - the remainder

.equ const,-0x33333333
#.equ const,0xcccccccd
.text
.align  2
.global divide_by_10_remainder
.type   divide_by_10_remainder, %function
divide_by_10_remainder:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     rsmall
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r3, r2          @ r3 = r2
        mov     r3, r3, asl #2  @ r3 = 4 * r3
        add     r3, r3, r2      @ r3 = r3 + r2
        mov     r3, r3, asl #1  @ r3 = 2 * r3
        rsb     r3, r3, r0      @ r3 = r0 - r3 = r0 - 10*int(r0/10)
        mov     r1, r3          @ the remainder
        mov     r0, r2          @ the dividend
        b       rlast
rsmall:
        mov     r1, r0
        mov     r0, 0
rlast:
        ldmfd   sp!, {pc}


# this subroutine divides the passed number by 10
# returns the dividend
#
# The const -0x33333333 is 0xcccccccd (2s complement)
# 0xccccccccc is 12/15th (0.8) of 0xffffffff and we use this as
# a multiplier, then shift right by 3 bits (divide by 8) to
```

```
# effect a multiplication by 0.1
#
# We multiply this number by 10 (multiply by 4, add 1 then multiply by 2)
#
# inputs
#   r0 - integer to divide
#
# outputs
#   r0 - the dividend

.align  2
.global divide_by_10
.type   divide_by_10, %function
divide_by_10:
        stmfd   sp!, {lr}
        cmp     r0, 10
        blt     small
        ldr     r1, =const
        umull   r2, r3, r1, r0
        mov     r2, r3, lsr #3  @ r2 = r3 / 8 == r0 / 10
        mov     r0, r2          @ the dividend
        b       last
small:
        mov     r0, 0
last:
        ldmfd   sp!, {pc}
```

copybytes.s

```
.syntax unified

# copybytes takes input pointer in r0, input length in r1 and writes to r2
ptr     .req r4
len     .req r5
.global copybytes
.type   copybytes, %function
copybytes:
        stmfd   sp!, {ptr, len, lr}
        mov     len, r1
        mov     ptr, r2
copybytesloopstart:
        ldrb    r3, [r0], 1
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     copybytesloopstart

        mov     r0, ptr
        mov     r1, len
        ldmfd   sp!, {ptr, len, pc}
```

clearbytes.s

```
.syntax unified

# clearbytes takes pointer in r0, input length in r1
# and sets all vector elements to 0

.global clearbytes
.type   clearbytes, %function
clearbytes:
        stmfd   sp!, {lr}
        mov     r3, 0
clearbytesloopstart:
        strb    r3, [r0], 1
        subs    r1, r1, 1
        bne     clearbytesloopstart

        ldmfd   sp!, {pc}
```

printbytes.s

```
.syntax unified

# printbytes takes input pointer in r0, input length in r1 and writes printable vector to r2 (with trailing null)
        .global printbytes
        .type   printbytes, %function
printbytes:
        stmfd   sp!, {lr}
printbytes_loopstart:
        ldrb    r3, [r0], 1
        add     r3, r3, 48
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     printbytes_loopstart

        mov     r3, 0
        strb    r3, [r2], 1
        ldmfd   sp!, {pc}
```

Description of problem

Let d(*n*) be defined as the sum of proper divisors of *n* (numbers less than *n* which divide evenly into *n*).
If d(*a*) = *b* and d(*b*) = *a*, where *a* â‰ *b*, then *a* and *b* are an amicable pair and each of *a* and *b* are called amicable numbers.

For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore d(220) = 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; so d(284) = 220.

Evaluate the sum of all the amicable numbers under 10000.

pe021.s

```
.syntax unified

.equ    SIZE, 10000
.equ    SIZEB, 40000

.align 4
```

```
aptr            .req r4
number          .req r4
sum             .req r5
tmp             .req r5
icount          .req r6
total           .req r7

.section .bss
.lcomm array,SIZEB

.section .rodata
resstring:
        .asciz "%d\n"

.text
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4-r8, lr}
        mov     icount, 0
        ldr     aptr, =array
array_loop:
        mov     r0, icount
        bl      sum_factors
        str     r0, [aptr], 4
        add     icount, icount, 1
        ldr     tmp, =SIZE
        cmp     icount, tmp
        blt     array_loop

        ldr     aptr, =array
        mov     tmp, 0
        ldr     icount, =SIZE
        mov     total, 0
ploop:
        ldr     r2, [aptr, tmp, lsl 2]
        cmp     r2, icount
        bge     pnext
        teq     tmp, r2
        beq     pnext
        ldr     r3, [aptr, r2, lsl 2]
        teq     tmp, r3
        bne     pnext
        add     total, total, tmp
pnext:
        add     tmp, tmp, 1
        cmp     tmp, icount
        bne     ploop
printme:
        mov     r1, total
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r8, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

.global sum_factors
.type   sum_factors, %function
sum_factors:
        stmfd   sp!, {r4-r6, lr}
        mov     number, r0
        mov     sum, 1
        mov     icount, 2
sf_loop:
        mul     r0, icount, icount
        cmp     r0, number
        bgt     sf_end
        mov     r0, number
        mov     r1, icount
        bl      divide
        teq     r1, 0
        bne     sf_next
        add     sum, sum, r0
        add     sum, sum, icount
sf_next:
        add     icount, icount, 1
        b       sf_loop
sf_end:
        mov     r0, sum
        ldmfd   sp!, {r4-r6, pc}
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

Using names.txt (right click and "Save Link/Target As..."), a 46K text file containing over five-thousand first names, begin by sorting it into alphabetical order. Then working out the alphabetical value for each name, multiply this value by its alphabetical position in the list to obtain a name score.

For example, when the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 938th name in the list. So, COLIN would obtain a score of 938 Ã— 53 = 49714.

What is the total of all the name scores in the file?

pe022.s

```
.syntax unified

.equ    comma, 44
.equ    ASCII, 64
```

```
.equ    NAMES, 5163
# from perl -ne '$count = $_ =~ tr/,/,/; print $count,"\n"' ../names.txt

.equ    SIZE, 46448
#from wc -c ../names.txt


.align 4

names_ptr       .req r4
count           .req r5
nstart          .req r6
tmp             .req r7
nsize           .req r7
sorted_ptr      .req r7
start_ptr       .req r8
size_ptr        .req r9
swapped         .req r10

.section .bss
.lcomm namestart,NAMES<<1       @ need 16 bit ints (half words) to handle 5163
.lcomm namesize,NAMES           @ none of the names are > 255 characters so use bytes
.lcomm sorted,NAMES<<1
.lcomm printname,63
# actually namesize is superfluous; we know the the size of i is from istart to (i+1)start-3? (3 from 2 " and 1 ,)

.section .rodata
res_string:
        .asciz "%d\n"
names:
.asciz "\"MARY\",\"PATRICIA\",\"LINDA\",\"BARBARA\",\"ELIZABETH\",\"JENNIFER\",\"MARIA\",\"SUSAN\",\"MARGARET\",\"DOROTHY\",\"LISA\",\"NANCY\",\"KAREN\",\"BETTY\",\"HELEN\",\"SANDRA\",\"DONNA\",\"CAROL\",\"RUTH\",\"SHARON\",\"MICHELLE\",\"LAURA\",\"SARAH\",\"KIMBERLY\",\"DEBORAH\",\"JESSICA\",\"SHIRLEY\",\"CYNTHIA\",\"ANGELA\",

.text
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4-r10, lr}

        bl      init_sorted
        bl      parse_names
        bl      vectored_bubblesort
        bl      compute_score
        mov     r1, r0
        ldr     r0, =res_string
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

# init_sorted initialises the sorted list to i
.type   init_sorted, %function
init_sorted:
        stmfd   sp!, {r5, r7, lr}
        ldr     sorted_ptr, =sorted
        mov     count, 0
        ldr     r3, =NAMES
siloop:
        strh    count, [sorted_ptr], 2
        add     count, count, 1
        cmp     count, r3
        bne     siloop
        ldmfd   sp!, {r5, r7, pc}

# parse_names parses the name string and populates namestart and namesize lists
.type   parse_names, %function
parse_names:
        stmfd   sp!, {r4-r10, lr}
        ldr     names_ptr, =names
        ldr     start_ptr, =namestart
        ldr     size_ptr, =namesize
        mov     count, 0
        mov     nstart, 1
        ldr     r3, =SIZE
iloop:
        ldrb    r0, [names_ptr], 1
        add     count, count, 1
        teq     r0, 0                   @ use .asciz for string so it is null-terminated
        beq     iloopend
        cmp     r0, #comma
        bne     iloop

        strh    nstart, [start_ptr], 2
        sub     nsize, count, nstart
        sub     nsize, nsize, 2
        strb    nsize, [size_ptr], 1
        add     nstart, count, 1

        cmp     count, r3
        blt     iloop
        beq     ilast
iloopend:
        strh    nstart, [start_ptr], 2
        sub     count, count, 2
        sub     nsize, count, nstart
        strb    nsize, [size_ptr]
ilast:
        ldmfd   sp!, {r4-r10, pc}


# vectored_bubblesort uses the indirection vector, sorted, and sorts the names by manipulating the contents of the vector
# we could use a different sorting algortithm, this is the simplest to implement but it is inefficient
.type   vectored_bubblesort, %function
vectored_bubblesort:
        stmfd   sp!, {r4-r10, lr}
        mov     swapped, 0
bubblestart:
        ldr     names_ptr, =names
        ldr     sorted_ptr, =sorted
        ldr     count, =NAMES
        sub     count, count, 1
bubbleloop:
        ldrh    r6, [sorted_ptr], 2
        ldr     start_ptr, =namestart
        add     start_ptr, start_ptr, r6, lsl 1
        ldr     size_ptr, =namesize
        add     size_ptr, size_ptr, r6
```

```
        ldrh    r0, [start_ptr]
        add     r0, r0, names_ptr
        ldrb    r1, [size_ptr]

        ldrh    r6, [sorted_ptr]
        ldr     start_ptr, =namestart
        add     start_ptr, start_ptr, r6, lsl 1
        ldr     size_ptr, =namesize
        add     size_ptr, size_ptr, r6

        ldrh    r2, [start_ptr]
        add     r2, r2, names_ptr
        ldrb    r3, [size_ptr]
        bl      compare

        cmp     r0, 1
        ldrheq  r0, [sorted_ptr, -2]
        ldrheq  r1, [sorted_ptr]
        strheq  r1, [sorted_ptr, -2]
        strheq  r0, [sorted_ptr]
        moveq   swapped, 1
        subs    count, count, 1
        bne     bubbleloop
        teq     swapped, 1
        mov     swapped, 0
        beq     bubblestart
        ldmfd   sp!, {r4-r10, pc}

# sumname takes start in r0, size in r1 and returns the sum of letters-ASCII in r0
.type   sumname, %function
sumname:
        stmfd   sp!, {lr}
        ldr     r2, =names
        add     r2, r2, r0
        mov     r0, 0
nextsumchar:
        ldrb    r3, [r2], 1
        sub     r3, r3, #ASCII
        add     r0, r0, r3
        subs    r1, r1, 1
        bne     nextsumchar
        ldmfd   sp!, {pc}

# compute_score computes the score from the sorted vector and returns it in r0
.type   compute_score, %function
compute_score:
        stmfd   sp!, {r4-r10, lr}
        mov     r4, 0
        mov     count, 0
        ldr     sorted_ptr, =sorted
        ldr     r10, =NAMES
cs_start:
        ldrh    r6, [sorted_ptr], 2
        ldr     start_ptr, =namestart
        add     start_ptr, start_ptr, r6, lsl 1
        ldr     size_ptr, =namesize
        add     size_ptr, size_ptr, r6
        ldrh    r0, [start_ptr]
        ldrb    r1, [size_ptr]
        bl      sumname
        mov     r6, r0
        add     count, count, 1
        mul     r6, r6, count
        add     r4, r4, r6
        cmp     count, r10
        blt     cs_start

        mov     r0, r4
        ldmfd   sp!, {r4-r10, pc}
```

compare.s

```
# compare implements the same functionality as the forth word of the
# same name.

# Compare the string specified by c-addr1 (r0) and u1 (r1) to the string
# specified by c-addr2 (r2) and u2 (r3) . The strings are compared, beginning
# at the given addresses, character by character up to the length of the
# shorter string, or until a difference is found. If both strings are the same
# up to the length of the shorter string, then the longer string is greater
# than the shorter string. n is -1 if the string specified by c-addr1 and u1
# is less than the string specified by c-addr2 and u2. n is zero if the
# strings are equal. n is 1 if the string specified by c-addr1 and u1 is
# greater than the string specified by c-addr2 and u2.

.syntax unified

.text

minlen  .req r4
val1    .req r5
val2    .req r6
count   .req r7

        .global compare
        .type   compare, %function
compare:
        stmfd   sp!, {r4-r8, lr}
        mov     minlen, r1
        cmp     minlen, r3
        movgt   minlen, r3
        mov     count, 0
loopstart:
        ldrb    val1, [r0, count]
        ldrb    val2, [r2, count]
        cmp     val1, val2
        movlt   r0, -1
        movgt   r0, 1
        bne     loopend
        add     count, count,1
        cmp     count, minlen
        bne     loopstart
        cmp     r1, r3
        moveq   r0, 0
        movlt   r0, -1
        movgt   r0, 1
loopend:
```

```
        ldmfd   sp!, {r4-r8, pc}
```

Description of problem

A perfect number is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of 28 would be 1 + 2 + 4 + 7 + 14 = 28, which means that 28 is a perfect number.

A number *n* is called deficient if the sum of its proper divisors is less than *n* and it is called abundant if this sum exceeds *n*.

As 12 is the smallest abundant number, 1 + 2 + 3 + 4 + 6 = 16, the smallest number that can be written as the sum of two abundant numbers is 24. By mathematical analysis, it can be shown that all integers greater than 28123 can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.

Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.

pe023.s

```
.syntax unified

.equ    datum_size, 2
.equ    SIZE, 28123

.align 4

aptr            .req r4
sfcount         .req r4
sum             .req r5
bptr            .req r5
number          .req r6
total           .req r7
last            .req r8
addi            .req r9
icount          .req r9
tmp             .req r10

.section .bss
.lcomm array,SIZE<<1    @ use 16-bit ints for the list
.lcomm bitarray,SIZE

.section .rodata
resstring:
        .asciz "%d\n"

.text
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4-r10, lr}
# store the abundant numbers to the vector array and set the elements corresponding
# to the sum of the factors in the bit vector abundantbit
        mov     icount, 1
        mov     number, 0
        ldr     aptr, =array
        ldr     bptr, =bitarray
        ldr     last, =SIZE
array_loop:
        mov     r0, icount
        bl      sum_factors

        cmp     icount, r0
        bge     array_next
        strhlt  icount, [aptr], #datum_size
        addlt   number, number, 1
        cmp     r0, last
        movlt   r1, 1
        strblt  r1, [bptr, icount]
array_next:
        add     icount, icount, 1
        cmp     icount, last
        blt     array_loop

        mov     total, 0
# add all of the integers until we reach the first abundant number
        ldr     aptr, =array
        ldrh    tmp, [aptr]
sloop:
        subs    tmp, tmp, 1
        add     total, total, tmp
        bne     sloop

        ldr     aptr, =array
        ldrh    r0, [aptr]      @ r0 is the index of the outer loop
ploop:
        ldr     aptr, =array
        ldr     bptr, =bitarray
        mov     addi, 1
        mov     r3, 0           @ r3 is the index of the inner loop
iloop:
        ldrh    tmp, [aptr], #datum_size
        cmp     r0, tmp
        blt     ilast
        sub     r1, r0, tmp
        ldrb    r2, [bptr, r1]
        teq     r2, 1
        moveq   addi, 0
        beq     ilast
        add     r3, r3, 1
        cmp     number, r3
        bne     iloop
ilast:
        teq     addi, 1
        addeq   total, total, r0
        add     r0, r0, 1
        cmp     r0, last
        bne     ploop
printme:
        mov     r1, total
        ldr     r0, =resstring  @ store address of start of string to r0
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4-r10, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

```
.global sum_factors
.type   sum_factors, %function
sum_factors:
        stmfd   sp!, {r4-r6, lr}
        mov     number, r0
        mov     sum, 1
        mov     sfcount, 2
sf_loop:
        mul     r0, sfcount, sfcount
        cmp     r0, number
        bgt     sf_end
        mov     r0, number
        mov     r1, sfcount
        bl      divide
        teq     r1, 0
        bne     sf_next
        add     sum, sum, r0
        cmp     r0, sfcount
        addne   sum, sum, sfcount
sf_next:
        add     sfcount, sfcount, 1
        b       sf_loop
sf_end:
        mov     r0, sum
        ldmfd   sp!, {r4-r6, pc}
```

divide.s

```
.syntax unified
# divide takes value in r0, divisor in r1 and returns dividend in r0 and modulus in r1
        .global divide
        .type   divide, %function
divide:
        stmfd   sp!, {lr}
# see http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043d/IHI0043D_rtabi.pdf
        bl      __aeabi_uidivmod
        ldmfd   sp!, {pc}
```

Description of problem

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3 and 4. If all of the permutations are listed numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:

012 021 102 120 201 210

What is the millionth lexicographic permutation of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9?

pe024.s

```
.syntax unified

# - see http://en.wikipedia.org/wiki/Factorial_number_system
# - and http://en.wikipedia.org/wiki/Permutation

.macro check_digit a
        ldr     r0, =vector
        mov     r1, VSIZE
        mov     r2, \a
        bl      contains
        teq     r0, 1
        bne     mloop
.endm

.macro copy_vector a b
        ldr     r0, =\a
        add     r0, r0, 1
        mov     r1, VSIZE
        ldr     r2, =\b
        bl      copybytes
.endm

# this macro lifted from test_add_digit_strings.s
# the usage of add_digit_strings is tricky because
# the fifth parameter must be passed on the stack

.macro add_strings a al b bl c
        stmfd   sp!, {r4}       @ stash r4 on the stack - we destroy it in add_digit_strings
        ldr     r0, =\c
        stmfd   sp!, {r0}       @ this is the fifth parameter for the subroutine
        ldr     r0, =\a
        ldr     r1, =\al
        ldr     r2, =\b
        ldr     r3, =\bl
        bl      add_digit_strings
        add     sp, sp, 4       @ revert sp to before (1)
        ldmfd   sp!, {r4}       @ and get stashed r4
.endm

.equ    datum_size, 2
.equ    SIZE, 2080
.equ    VSIZE, 10

.align 4

icount          .req r4

.section .data
.align  2
vector:
.byte 2, 7, 8, 0, 0, 0, 0, 0, 0, 0
increment:
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 1

.section .bss
.lcomm output, VSIZE
.lcomm print_vector, VSIZE

.section .rodata
outstring:
.asciz "%s\n"

.text
```

```
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4, lr}
        ldr     icount, =SIZE
mloop:
        add_strings vector 10 increment 10 output
        copy_vector output vector
        check_digit 0
        check_digit 1
        check_digit 2
        check_digit 3
        check_digit 4
        check_digit 5
        check_digit 6
        check_digit 7
        check_digit 8
        check_digit 9
        subs    icount, icount, 1
        bne     mloop
printme:
        ldr     r0, =vector
        mov     r1, 10
        ldr     r2, =print_vector
        bl      printbytes

        ldr     r1, =print_vector
        ldr     r0, =outstring
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux

# contains takes a pointer to a byte vector in r0, and a size in r1 and a scalar in r2
# it returns 1 in r0 if the vector contains the scalar and 0 otherwise
.global contains
.type   contains, %function
contains:
        stmfd   sp!, {r4, lr}
        mov     r4, r0
        mov     r0, 0
contains_start:
        ldrb    r3, [r4], 1
        cmp     r3, r2
        moveq   r0, 1
        beq     contains_end
        subs    r1, r1, 1
        bne     contains_start
contains_end:
        ldmfd   sp!, {r4, pc}
```

add_digit_strings.s

```
.syntax unified

# see usage in test_add_digit_strings.s - it requires
# the fifth parameter to be passed on the stack

.equ datum_size, 1

.text
optr            .req r4
sptr            .req r5
lptr            .req r6
scell           .req r7
lcell           .req r8
carry           .req r9
ltmp            .req r9
counter         .req r10
ptmp            .req r10

# this subroutine adds the byte array at r0, length r1
# to the byte array at r2, length r3. The data is output
# to the pointer passed as r4.
#
# inputs
#   r0 - pointer to input1 vector
#   r1 - length of input1 vector
#   r2 - pointer to input2 vector
#   r3 - length of input2 vector
#   r4 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

.text
.align  2

.global add_digit_strings
.type add_digit_strings, %function
add_digit_strings:
        stmfd   sp!, {r9-r10, lr}
        cmp     r3, r1
        movlt   ptmp, r0
        movlt   r0, r2
        movlt   r2, ptmp
        movlt   ltmp, r1
        movlt   r1, r3
        movlt   r3, ltmp
        bl      add_strings_short_to_long
        ldmfd   sp!, {r9-r10, pc}

# this subroutine adds the short byte array at r0, length r1
# to the byte array at r2, length r3.
#
# inputs
#   r0 - pointer to short vector
#   r1 - length of short vector
#   r2 - pointer to long vector
#   r3 - length of long vector
#   r4 - pointer to output vector
#
# outputs
```

```
#   r0 - pointer to output vector
#   r1 - length of output vector

.global add_strings_short_to_long
.type add_strings_short_to_long, %function
add_strings_short_to_long:
        stmfd   sp!, {r5-r10, lr} @ 7 longs
        ldr     optr, [sp, #40]
        mov     sptr, r0
        add     sptr, sptr, r1
        sub     sptr, sptr, 1

        mov     lptr, r2
        add     lptr, lptr, r3
        sub     lptr, lptr, 1

        add     optr, optr, r3

        mov     carry, 0
        mov     counter, r1
sstart:
        ldrb    scell, [sptr], -1
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, scell
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     sstart

        mov     counter, r3
        subs    counter, counter, r1
        beq     asstl_last
lstart:
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     lstart

asstl_last:
        cmp     carry, 1
        strbeq  carry, [optr], -1

        add     r0, optr, 1
        add     r1, r3, carry
        ldmfd   sp!, {r5-r10, pc}
```

printbytes.s

```
.syntax unified

# printbytes takes input pointer in r0, input length in r1 and writes printable vector to r2 (with trailing null)
        .global printbytes
        .type   printbytes, %function
printbytes:
        stmfd   sp!, {lr}
printbytes_loopstart:
        ldrb    r3, [r0], 1
        add     r3, r3, 48
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     printbytes_loopstart

        mov     r3, 0
        strb    r3, [r2], 1
        ldmfd   sp!, {pc}
```

copybytes.s

```
.syntax unified

# copybytes takes input pointer in r0, input length in r1 and writes to r2
ptr     .req r4
len     .req r5
.global copybytes
.type   copybytes, %function
copybytes:
        stmfd   sp!, {ptr, len, lr}
        mov     len, r1
        mov     ptr, r2
copybytesloopstart:
        ldrb    r3, [r0], 1
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     copybytesloopstart

        mov     r0, ptr
        mov     r1, len
        ldmfd   sp!, {ptr, len, pc}
```

Description of problem

The Fibonacci sequence is defined by the recurrence relation:

$F_n = F_{n-1} + F_{n-2}$, where $F_1 = 1$ and $F_2 = 1$.

Hence the first 12 terms will be:

$F_1 = 1$
$F_2 = 1$
$F_3 = 2$

$F_4 = 3$
$F_5 = 5$
$F_6 = 8$
$F_7 = 13$
$F_8 = 21$
$F_9 = 34$
$F_{10} = 55$
$F_{11} = 89$
$F_{12} = 144$

The 12th term, F_{12}, is the first term to contain three digits.

What is the index of the first term in the Fibonacci sequence to contain 1000 digits?

pe025.s

```
.syntax unified

.macro copy_vector a b
        ldr     r0, =\a
        add     r0, r0, 1
        mov     r1, VSIZE
        ldr     r2, =\b
        bl      copybytes
.endm

# this macro lifted from test_add_digit_strings.s
# the usage of add_digit_strings is tricky because
# the fifth parameter must be passed on the stack

.macro add_strings a al b bl c
        stmfd   sp!, {r4}       @ stash r4 on the stack - we destroy it in add_digit_strings
        ldr     r0, =\c
        stmfd   sp!, {r0}       @ this is the fifth parameter for the subroutine
        ldr     r0, =\a
        ldr     r1, =\al
        ldr     r2, =\b
        ldr     r3, =\bl
        bl      add_digit_strings
        add     sp, sp, 4       @ revert sp to before (1)
        ldmfd   sp!, {r4}       @ and get stashed r4
.endm

.macro add_and_test ivector ovector
        add_strings \ovector VSIZE \ivector VSIZE output
        copy_vector output \ovector
        add     icount, icount, 1
        ldr     vptr, =\ovector
        ldrb    tmp, [vptr]
        teq     tmp, 0
        bne     printme
.endm

.equ    VSIZE, 1000

.align 4

vptr            .req r2
tmp             .req r3
icount          .req r4

.section .data
.align  2
vector1:
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
vector2:
.byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

.section .bss
.lcomm output, VSIZE

.section .rodata
resstring:
.asciz "%d\n"

.text
.align  2
.global main
.type   main, %function
main:
        stmfd   sp!, {r4, lr}
        mov     icount, 2
mloop:
        add_and_test vector1 vector2
        add_and_test vector2 vector1
        b       mloop
printme:
        mov     r1, icount
        ldr     r0, =resstring
        bl      printf

        mov     r0, 0
        ldmfd   sp!, {r4, pc}
        mov     r7, 1           @ set r7 to 1 - the syscall for exit
        swi     0               @ then invoke the syscall from linux
```

add_digit_strings.s

```
.syntax unified

# see usage in test_add_digit_strings.s - it requires
# the fifth parameter to be passed on the stack

.equ datum_size, 1

.text
optr            .req r4
sptr            .req r5
```

```
lptr            .req r6
scell           .req r7
lcell           .req r8
carry           .req r9
ltmp            .req r9
counter         .req r10
ptmp            .req r10

# this subroutine adds the byte array at r0, length r1
# to the byte array at r2, length r3. The data is output
# to the pointer passed as r4.
#
# inputs
#   r0 - pointer to input1 vector
#   r1 - length of input1 vector
#   r2 - pointer to input2 vector
#   r3 - length of input2 vector
#   r4 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

.text
.align  2

.global add_digit_strings
.type add_digit_strings, %function
add_digit_strings:
        stmfd   sp!, {r9-r10, lr}
        cmp     r3, r1
        movlt   ptmp, r0
        movlt   r0, r2
        movlt   r2, ptmp
        movlt   ltmp, r1
        movlt   r1, r3
        movlt   r3, ltmp
        bl      add_strings_short_to_long
        ldmfd   sp!, {r9-r10, pc}

# this subroutine adds the short byte array at r0, length r1
# to the byte array at r2, length r3.
#
# inputs
#   r0 - pointer to short vector
#   r1 - length of short vector
#   r2 - pointer to long vector
#   r3 - length of long vector
#   r4 - pointer to output vector
#
# outputs
#   r0 - pointer to output vector
#   r1 - length of output vector

.global add_strings_short_to_long
.type add_strings_short_to_long, %function
add_strings_short_to_long:
        stmfd   sp!, {r5-r10, lr} @ 7 longs
        ldr     optr, [sp, #40]
        mov     sptr, r0
        add     sptr, sptr, r1
        sub     sptr, sptr, 1

        mov     lptr, r2
        add     lptr, lptr, r3
        sub     lptr, lptr, 1

        add     optr, optr, r3

        mov     carry, 0
        mov     counter, r1
sstart:
        ldrb    scell, [sptr], -1
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, scell
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     sstart

        mov     counter, r3
        subs    counter, counter, r1
        beq     asstl_last
lstart:
        ldrb    lcell, [lptr], -1
        add     lcell, lcell, carry
        mov     carry, 0
        cmp     lcell, 10
        movge   carry, 1
        subge   lcell, lcell, 10
        strb    lcell, [optr], -1
        subs    counter, counter, 1
        bne     lstart

asstl_last:
        cmp     carry, 1
        strbeq  carry, [optr], -1

        add     r0, optr, 1
        add     r1, r3, carry
        ldmfd   sp!, {r5-r10, pc}
```

copybytes.s

```
.syntax unified

# copybytes takes input pointer in r0, input length in r1 and writes to r2
ptr     .req r4
len     .req r5
.global copybytes
.type   copybytes, %function
copybytes:
        stmfd   sp!, {ptr, len, lr}
```

```
        mov     len, r1
        mov     ptr, r2
copybytesloopstart:
        ldrb    r3, [r0], 1
        strb    r3, [r2], 1
        subs    r1, r1, 1
        bne     copybytesloopstart

        mov     r0, ptr
        mov     r1, len
        ldmfd   sp!, {ptr, len, pc}
```