

Solving the Google "prime number in e" challenge

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-07/0925.html>

From: John Dahlman (jrdahlman2_at_netscape.net)

Date: 07/22/04

Date: 22 Jul 2004 09:49:06 -0700

(or, "Of Polish Wizards and Russian Peasants")

Over a week ago I saw this in the local paper:

JOB HUNTERS, GOOGLE THIS

Source: MICHAEL BAZELEY, Mercury News

"Hoping to tickle the minds of Silicon Valley's brightest engineers, the simple black-on-white billboard by a famous Internet company displays nothing more than a mathematical puzzle to drivers on Highway 101:

{first 10-digit prime found in consecutive digits of e}.com

The answer leads to a Web site, where another math question awaits. Those who solve that one get the prize: the chance to submit their resume to Google Labs through a special e-mail address...."

Published on July 10, 2004, Page 1C, San Jose Mercury News (CA)

Well I like the job that I have, but it did look like an interesting mathematical challenge. After promising myself I wouldn't touch it, I ended up spending all week writing an assembly program to get the answer. (I'll bet serious Google-hunters wouldn't even bother—they'd just Google for the first person to publically beat it.)

Here you'll find a RosAsm assembly program that computes the decimal digits of e (2.718...), makes 10 digit numbers out of that string, and then tests each as a prime number. You won't find the answer—you'll have to run the program for that. I used RosAsm because that's what's in front of me, but I hope it's clear enough to be easily translated to other dialects. (No endorsement or criticism of any particular assembler is intended or implied.)

And how did I know how to solve the problem? Simple: I used a time-honored mathematical method: I copied the ideas out of a book.

1. The fractional digits of "e": "The Polish Wizard"

alt.lang.asm: Solving the Google "prime number in e" challenge

Hunting for the digits of e isn't as famous as, say, pi. But it has been done before, in assembly language no less. In the June 1981 issue of Byte magazine there was an article for "computing e to 116,000 places" by none other than the Woz himself, Steve Wozniak. (It's also in the 1994 edition of "The Best of Byte", which is where I found it.) He used every 6502 trick he knew, and it only took 4 solid days to fill an Apple II with 47K of the binary fraction of e.

Ignoring the "2.", the fractional part of e is defined as:

$$E = 1/2! + 1/3! + \dots 1/N!$$

where you make N big enough for number of digits you want. You have two multi-kilobyte-long numbers you have to calculate with: e and N!

Wozniak realized it could also be calculated this way:

$$E = ((((((1 / N)+1) / (N-1))+1) / \dots +1) / 2)$$

or add 1, divide, add 1, divide... decementing N each time, down to 2. (Very clever. I'd never have seen that in a million years.) Now you only have one multi-kilobyte number to deal with, E, that you divide by N, a regular integer.

To do that, the Woz algorithm runs through a long memory array (E) one byte at a time. (It "need not be initialized, since it will all be reduced to insignificance when divided by N!") N is the 16-bit divisor. A 16-bit carried remainder (A2A1) is combined with a byte from the E array (A0) to form a 24-bit dividend (A2A1A0) that you divide by N. The LSB (A0) of the quotient goes back into E, while the remainder becomes the A2A1 bytes of a new dividend. After going through the whole E array, N is decremented, and start over again at the beginning of E for the next round of long division. (The "add 1" part is handled by presetting the remainder to 1 at the beginning of each round.)

My program creates a 1024-byte array for the e digits, although you can make the article's 14K version by using the commented-out lines. (Turns out you don't need to go that far.) Old hands at assembly will find it offensively UN-optimized. I used the string instructions because they were familiar, and it works a-byte-at-a-time because that's how the original worked, and I wanted to guarantee an accurate answer. (And any bigger than a byte and we run into Intel's big-endian—or is it little-endian?—backwardness.) Confirmed the result against the article's check bytes: first byte should be B7h, and the 1024th should be 24h.

The result is the fractional digits of e in a *binary* (or hexadecimal) form. The whole thing has to be converted to decimal.

2. From binary to decimal.

alt.lang.asm: Solving the Google "prime number in e" challenge

The same Byte article shows how to get the decimal digits: repeatedly multiply the whole shebang by 10, picking off the digits one at a time. (Integers are converted to decimal by division, but this is a binary *fraction*.) Since we have more room than a 48K Apple II, I make a second array to hold them. Once again, I went slow—but—careful: a byte and a digit at a time. (Stored as unpacked decimal: just add 48 to each byte for quick printing.) Prints the lines in the same format as the article—all match. If you're curious, the first line is:

```
(2.)71828 18284 59045 23536 02874 71352 66249 77572 47093 69995 95749 66967
```

3. Lots of ten-digit numbers.

Ten decimal digits holds up to 9,999,999,999 or 10 billion. (American billion.) Since over half of those numbers won't fit in a 32-bit integer, we switch to the FPU. Fortunately, the FPU can load BCD. (Unfortunately, only packed BCD and backwards. Another subroutine to write....)

4. Testing for primes: Russian Peasant method -- FAILED

I *thought* I knew how to check each ten-digit number for prime without factoring, assuming factoring would be too long. There's a probabilistic test for N for prime (if $\text{Random} \wedge (N-1) \bmod N < 1$, it's not), using the "Russian Peasant method" for modular powers. It worked... if the number was less than 9 digits long. The ten-digit ones failed on this: $X * X \bmod N$. An FPU register wouldn't hold the *square* of a ten-digit number without rounding.

5. Testing for primes: brute-force factoring.

After spending half a week discovering the above problem, I finally said heck with it and threw in a quick brute-force factorizer test. No primes; just tries every odd integer up to the square-root of the number. (Though if it does find a factor it divides it and starts again.) If no factors found, it's a prime. Figured it would take forever, but I was getting desperate. Guess what? Even on the dinky little 75mhz Pentium in my dinky little Libretto, it found the right prime immediately!

That'll teach me to be too clever.

The number is one right above the words "It's a prime." If you add ".com" at the end put it in a browser, yes, you get a very plain website. There's another puzzle there, if you want to tackle it. I quit.

Comments welcome, optimizations appreciated, criticism of the algorithms expected, flames & further rounds of the Death Match will be ignored.

John Dahlman

"All I know of computer science I learned from 'Byte.'
Unfortunately, my subscription ran out in 1990."

alt.lang.asm: Solving the Google "prime number in e" challenge

Note on the program:

This is not a "windowed" program, or even a console one. Besides RosAsm, I like to use Eric ("RobotBob") Asbell's QDEBUG.EXE window for output, called through QDEBUG.DLL. They're not part of the standard RosAsm set, but can be found easily at his website:

www.quanta-it.com/easbell/examples.php

Just a convenient way to print to a window; feel free to replace with console or other equivalents. The 3 routines I use are:

"Outputhex" print register or memory variable in hex.

"OutputString" print contents of string variable

"OutputStringLiteral" (equated to "Print") print following string

Start RosAsm, and name a new program.

Paste below directly into RosAsm's window, and compile (F5) or run (F6)

```
;e.exe: solves the Google prime-digits-of-e challenge.
;requires RobotBob's QDEBUG.DLL & QDEBUG.EXE for output

;the macros everyone uses
[push | push #1 | #+1]
[pop | pop #1 | #+1]

[call | push #L>2 | call #1]

;QDEBUG's macros
;note: all but StringLiteral print the "source = " plus result
[OutPutDec | &8=#1 | { &0: B$ '&8' 0 } | call 'QDEBUG.OutPutDec' #1 &0]
[OutPutHex | &8=#1 | { &0: B$ '&8' 0 } | call 'QDEBUG.OutPutHex' #1 &0]
[OutPutString | &8=#1 | { &0: B$ '&8' 0 } | call 'QDEBUG.OutPutString' #1 &0]
[OutPutStringLiteral | &8=#1 | { &0: B$ &8 0 } |
    call 'QDEBUG.OutPutStringSimple' &0 ]
[print OutPutStringLiteral] ;less typing

Main:
    print "The Woz calculation for the digits of e"
    print ""

[efrac: B$ 0 #1024] ;holds the digits of e
;[efrac: B$ ? #34524] ;(use for 14K version)

;eax is A2A1A0 3-byte dividend
;bx is N 2-byte divisor

    mov ebx 1000 ;start Divisor at 1000 for 1440 digits
    ;mov ebx 9720 ;start Divisor at 9720 for 34524 digits
```

alt.lang.asm: Solving the Google "prime number in e" challenge

```
nxtdivsr:
    mov esi efrac ;EE is readbyte's counter: reset to top
    mov edi efrac ;FF is appendbyte's counter: also reset
    mov edx 1 ;remainder is 1: add 1 each time
    mov ecx 1024 ;e is 1024 bytes long
F0:
    mov eax edx | mov edx 0 ;old remainder to be divided
    shl eax 8 ;shift remainder...
    lodsb ; ... and add byte from e
    div ebx ;divide by Divisor
    stosb ;quotient byte back to e
loop F0<
dec ebx
cmp ebx 2 | jnae O1> | jmp nxtdivsr ;2 is last divisor
```

```
O1:

    print "Calculation finished."
    print ""
    print "Check digits (hex)"
    print "first byte:"
    mov eax 0
    mov al B$efrac
    outputhex eax
    print "1024st byte:"
    mov eax 0
    mov al B$efrac+1023
    outputhex eax
```

```
[edec: B$ 0 #2000] ;make bigger for more digits
[ten: B$ 10]
    print ""
    print "Decimal digits:"
    mov ebx 0
    std ;going backwards: from LSB to MSB
```

```
nxtdec:
    mov esi efrac+1023 ;reset both to end
    mov edi efrac+1023
    mov eax 0 ;clear all A
    mov edx 0 ;clear carry
    mov ecx 1024 ;e is 1024 bytes long
F0:
    lodsb ;get LSB
    mul B$ten ;mult by 10
    add AX DX ;plus any OLD carry
    mov DL AH ;carry for NEXT time
    stosb ;new LSB byte back in
loop F0<
mov B$edec+ebx DL ;decimal digit ready
inc ebx
cmp ebx 1024 | jnb O1> | jmp nxtdec
```

```
O1:
```

alt.lang.asm: Solving the Google "prime number in e" challenge

```
cld ;don't forget

[e: B$ 0 #80] ;printing the digits on 80-char lines
print " 2."
mov esi edec
mov edx 1
newline:
mov ebx 1
mov edi e
next5:
mov ecx 5
F0:
lodsb
add AL 48
stosb
loop F0<
mov al 32 | stosb ;space every 5 digits
inc ebx
cmp ebx 12 | jnbe O1> | jmp next5 ;12 5-digit-groups per line
O1:
outputstring e
inc edx
cmp edx 18 | jnbe O2> | jmp newline ;18 lines for 1024 digits
O2:
print ""

print ""
print "ten-digit numbers from e:"

mov ebx 0
nxtten:
inc ebx
mov esi edec
add esi ebx ;decimal digits shifted by B
mov DL B$esi+9 ;get LSB
bt edx 0 ;check last bit of last byte for odd #
jnc nxtten ;skip even numbers
cmp B$esi 0 ;check if first digit is 0
jz nxtten ;skip if less than 10 digits
call printnextten
call loadBCD
;call primetest ;didn't work
call factorize
cmp ebx 100 | jnb O1> | jmp nxtten ;100 numbers tried
O1:

call 'KERNEL32.ExitProcess' 0
ret
```

;subroutines

alt.lang.asm: Solving the Google "prime number in e" challenge

printnextten:

[tendig: B\$ 0 #11] ;for printing digits (the 11th is 0 to end string)

```
    push esi
    mov edi tendig
    mov ecx 10
F0:    lodsb
        add AL 48
        stosb
        loop F0<
        outputstring tendig
    pop esi
ret
```

loadBCD: ;into FPU

[BCD: B\$ 0 #10]

```
    push esi
    mov edi BCD+4 ;5 bytes = 10 packed BCD digits
    mov ecx 5
F0:    lodsb ;first nibble
        shl AL 4 | mov DL AL ;shift & hold somewhere
        lodsb ;next nibble
        or AL DL ;byte = 2 nibbles together
        std ;while e digit go forward...
        stosb
        cld ;..the BCD number goes backwards
        loop F0<
        fbld T$BCD
        ;stop
        ;outputreal st0
    pop esi
ret
```

factorize:

[startfact: T\$ 3 two: T\$ 2]

```
;Crude brute-force factorizer
;Factors odd integer: call with N in ST0
;Tries all odd factors F from 3 up until past SQRT(N)
;Empties stack when done.
;No return value: prints "prime" if one found.
;Not robust: fails on evens, negatives, and non-integers
```

```
    fld T$startfact
    fxch st0 st1 ;ST0 = N, ST1 = F
    fld st0
    fsqrt ;ST0 = SQRT(N), ST1 = N (oldN), ST2 = F
    fld st1 ;ST0 = N, ST1 = SQRT(N), ST2 = oldN, ST3 = F
L0:
```

alt.lang.asm: Solving the Google "prime number in e" challenge

```
fld st3 ;F current factor
fld st1 ;N current number to test
fprem ;is F a factor in N? N mod F. 0 = yes
fst
fstsw ax | sahf
fstp st0 | fstp st0 ;unstack, unstack
jne L1>
    ;It's a factor. Divide. Will it work again?
fdiv st0 st3 ;N = N / F
fld1 | fcomp ;divided down to 1 (the end)?
fstsw ax | sahf
je L2> ;if yes, we're done.
fst st1 ;ST0 = N/F, ST1 = N/F, ST2 = oldN, ST3 = F
fsqrt ;ST0 = SQRT(N/F), ST1 = N/F, ST2 = oldN, ST3 = F
fxch st0 st1 ;ST0 = N/F, ST1 = SQRT(N/F), ST2 = oldN, ST3 = F
jmp L0<
L1:
    ;not a factor. Try next one.
fld T$two
faddp st0 st4 ;ST0 = N, ST1 = SQRT(N), ST2 = oldN, ST3 = F+2
fxch st3 | fcom ;if F+2 > SQRT(N)...
fxch st3 | fstsw ax | sahf
jbe L0< ;...we're done
L2:
    ;out
fstp st1 ;ST0 = N, ST1 = oldN, ST2 = F
ffree st2 ;ST0 = N, ST1 = oldN
fcompp
fstsw ax | sahf
jne L3> ;if N was never divided, it's a prime
    print "It's a prime"
L3:
    ;we're done

ret
```