Machine Language Versus Basic Prime Number Generation

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

Editor's Note:
Watch your local dealer for Marvin L. De Jong's new book: ***Programming and Interfacing the 6502.*** *Due in January, the 450 page work is expected to sell for $11.95. Publisher:*
Howard W. Sams Co.
Indianapolis, IN.

The attached program will calculate prime numbers of the form $2^N - 1$, for poster and/or prime number enthusiasts who also read COMPUTE. It was motivated by one of my students who was searching for perfect numbers (numbers whose factors add to give the number itself). The student wrote a BASIC program for an APPLE, and the program would calculate and print 2^{10000}. It took 11 hours to do this. Thinking that perhaps the same thing could be done in machine language, I wrote the program given here with only minor modifications. It calculated 2^{10000} in 11 minutes, illustrating the advantage in speed that machine language offers for certain tasks.

The program listed here calculates and prints $2^{11213} - 1$, a number that is known to be prime. With a little more memory space than the 4K on my AIM 65, one could calculate and print the largest known prime (as of this writing) number, namely $2^{44497} - 1$. The number of digits in a number of the form 2^N can be shown to be $1 + N\log 2$. In the program given we calculate 2^{11213} giving the number of digits as $1 + 11213\text{Log}_{10}(2)$ = 3376 digits. The number 2^{44497} requires 13395 digits. Each memory location can store two BCD digits, so 2^{11213}, requires 1688 or about 2K locations in memory.

Some notes on the program follow. We allocated locations $0400 to $0FFF to hold the number. This is many more locations than are required to find 2^{11213}, but the program was used to find some larger powers of two also. First, the locations that are to contain the number are cleared to zero. This occurs in instructions $0208 to $0220. The indirect indexed addressing mode is used to reference the memory locations to be cleared. The address of this table is stored in $0004 and $0005. Next, a one is stored in the lowest address of the table. This number is doubled 11213 times giving 2^{11213}. The locations $0000, $0001, and $0002 keep track of the number of doubling times. In the instructions located from $0261 to $0272 this number is tested to see if it has reached 11213. Finally, one is subtracted from the number and it is printed by calling an AIM 65 subroutine at $F000. Owners of other systems can simply use their own output subroutine. It should also be clear from this explanation and the program comments what locations in the program must be modified to handle other numbers of the form 2^N.

There is really no practical use for the program or the output. However, prime numbers and perfect numbers have been of considerable interest to mathematicians for centuries. Perhaps some 6502 user will discover an even larger prime number than 2^{44497}, but don't underestimate the task.

P.S. A lot of leading zeros get printed before the number starts.

```
$0208  A9 00      START   LDA $00          Load pointers to number table.
 020A  85 04              STA TABLE
 020C  A9 04              LDA $04
 020E  85 05              STA TABLE+1
 0210  A0 00              LDY $00          Initialize Y index to zero to
 0212  A9 00      NEXT    LDA $00          clear all table locations to zero.
 0214  91 04      LOOP    STA (TABLE),Y    Put zero in each table location.
 0216  C8                 INY              Increment Y to fill page with zeros.
 0217  D0 FB              BNE LOOP
 0219  E6 05              INC TABLE+1      Go to the next page in the table.
 021B  A5 05              LDA TABLE+1      Are all the pages completed?
 021D  C9 10              CMP $10
 021F  90 F1              BCC NEXT         No. Then fill another page.
 0221  A9 04              LDA $04          Yes.  Reset pointers to the base
 0223  85 05              STA TABLE+1      address of the table.
 0225  F8                 SED              All subsequent additions will be
 0226  A9 01              LDA $01          in decimal.
 0228  8D 00 04           STA TABLO        Start with one in lowest digit of
 022B  A9 00              LDA $00          the table.
 022D  85 00              STA LO           Initialize the addition counter to
 022F  85 01              STA MID          zero;  three locations ($00,$01,$02)
```

```
0231  85 02        STA HI          in page zero.
0233  18     COUNT  CLC             Clear carry for additions.
0234  A9 01        LDA $01         Increment the addition counter,
0236  65 00        ADC LO          LO, MID, and HI each time the number
0238  85 00        STA LO          is added to itself.
023A  A5 01        LDA MID         Carry from LO addition into MID.
023C  69 00        ADC $00
023E  85 01        STA MID         Result into MID.
0240  A5 02        LDA HI          Carry from MID addition into HI.
0242  69 00        ADC $00
0244  85 02        STA HI          Result into HI.
0246  18           CLC             Clear carry for adding THE NUMBER.
0247  B1 04  PAGAD  LDA (TABLE),Y  Get a piece of THE NUMBER.
0249  71 04        ADC (TABLE),Y   Add it to itself.
024B  91 04        STA (TABLE),Y   Store THE NUMBER.
024D  C8           INY             Increment Y to repeat the addition
024E  D0 F7        BNE PAGAD       for an entire page of memory.
0250  E6 05        INC TABLE+1     Increment the page number.
0252  08           PHP             Store P to keep track of any carry.
0253  A5 05        LDA TABLE+1     Have we finished adding the entire
0255  C9 10        CMP $10         table?
0257  B0 04        BCS DOWN        Yes. Then check to see if we have
0259  28           PLP             added enough times. No. Add more.
025A  4C 47 02     JMP PAGAD
025D  A9 04  DOWN  LDA $04         Reset table pointer.
025F  85 05        STA TABLE+1
0261  A5 00        LDA LO          Check add counter. Is it equal to
0263  C9 13        CMP $13         011213?
0265  D0 CC        BNE COUNT
0267  A5 01        LDA MID
0269  C9 12        CMP $12
026B  D0 C6        BNE COUNT
026D  A5 02        LDA HI
026F  C9 01        CMP $01
0271  90 C0        BCC COUNT
$0273 18           CLC             Subtract one from 2^11213 to get
0274  AD 00 04     LDA TABLO       THE PRIME NUMBER.
0277  E9 00        SBC $00
0279  8D 00 04     STA TABLO
027C  A9 0F        LDA $0F         Point to the top of the table to
027E  85 05        STA TABLE+1     read THE PRIME NUMBER out from
0280  A0 FF  UP    LDY $FF         the most-significant digit to the
0282  B1 04  THERE LDA (TABLE),Y   least-significant digit.
0284  A2 FE        LDX $FE         Convert the BCD digits to ASCII.
0286  48           PHA             Save two digits on the stack.
0287  4A           LSR A           Get the most-significant nibble.
0288  4A           LSR A           Move it into the low-order nibble.
0289  4A           LSR A
028A  4A           LSR A
028B  18     HERE  CLC
028C  69 30        ADC $30         Here we have an ASCII digit so
028E  20 00 F0     JSR PRINT       jump to the output routine.
0291  E8           INX             Do we need to get another digit?
0292  F0 06        BEQ AHED        No.
0294  68           PLA             Yes. Get the digits.
0295  29 0F        AND $0F         Mask the high-order nibble.
0297  4C 8B 02     JMP HERE        Convert it to ASCII.
029A  88     AHED  DEY             Get some more of the number from
029B  C0 FF        CPY $FF         the same page.
029D  D0 E3        BNE THERE
029F  C6 05        DEC TABLE+1     Change pages.
02A1  A5 05        LDA TABLE+1
02A3  C9 04        CMP $04
02A5  B0 D9        BCS UP
02A7  00           BRK             Back to the monitor.
```