intel®

AB-34

APPLICATION BRIEF

Integer Square Root Routine for the 8096

LIONEL SMITH
ECO APPLICATIONS ENGINEER

April 1989

Order Number: 270523-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

INTEGER SQUARE ROOT ROUTINE FOR THE 8096

CONTENTS

This Application Brief presents an example of calculating the square root of a 32-bit signed integer.

Theory

Newton showed that the square root can be calculated by repeating the approximation:

Xnew = (R/Xold + Xold)/2 ; Xold = Xnew

where: R is the radicand

Xold is the current approximation of the square root

Xnew is the new approximation

until you get an answer you like. A common technique for deciding whether or not you like the answer is to loop on the approximation until Xnew stops changing. If you are dealing with real (floating point) numbers this technique can sometimes get you in trouble because it's possible to hang up in the loop with Xnew alternating between two values. This is not the case with integers. As an example of how it all works, consider taking the square root of 37 with an initial guess (Xold) of 1:

Xnew = (37/1 + 1)/2 = 19; Xold = 19

Xnew = (37/19 + 19)/2 = 10; Xold = 10

Xnew = (37/10 + 10)/2 = 6; Xold = 6

Xnew = (37/6 + 6)/2 = 6; Xold = 6 – done!

Note that in integer arithmetic the remainder of a division is ignored and the square root of a number is floored (i.e. the square root is the largest integer which, when multiplied by itself, is less than or equal to the radicand).

Practice

The only significant problem in implementing the square root calculation using this algorithm is that the division of R by Xold could easily be a 32 by 32 divide if R is a 32 bit integer. This is ok if you happen to have a 32 by 32 divide instruction, but most 16-bit machines (including the 8096) only provide a 32 by 16 divide. However, a little bit of creative laziness will allow us to squeeze by using the 32 by 16 bit divide on the 8096.

The largest positive integer you can represent with a 32-bit two's complement number is 07fff5ffffh, or 2,147,483,647. The square root of this number is 0b504h, or 46,340. The largest square root that we can generate from a 32-bit radicand can be represented in 16-bits. If we are careful in picking our initial Xold we can do all of the divisions with the 32 by 16 divide instruction we have available. Picking the largest possible 16-bit number (0ffffh) will always work although it may slow the calculation down by requiring too many iterations to arrive at the correct result. The algorithm below takes a slightly more intelligent approach. It uses the normalize instruction to figure out how many leading zeros the 32-bit radicand has and picks an initial Xold based on this information. If there are 16 or more leading zeros then the radicand is less than 16 bits so an initial Xold of 0ffh is chosen. If the radicand is more than 16 bits then the initial Xold is calculated by shifting the value 0ffffh by half as many places as there were leading zeros in the radicand. To give credit where credit is due, I first saw this 'trick" in the January 1986 issue of Dr. Dobbs's Journal in a letter from Michael Barr of McGill University.

The routine was timed in a 12.0 Mhz 8096 as it calculated the square roots of all positive 32-bit numbers, the following numbers include the CALL and return sequence and were measured using Timer 1 of the 8096.

Minimum Execution Time:	24 microseconds
Maximum Execution Time:	236 microseconds
Average Execution Time:	102 microseconds

Comments

The program module which follows is part of a collection of routines which perform integer and real arithmetic on a software implemented tagged stack. The top element of the stack is call TOS and is in fixed locations in the register file. Since the square root operation only involves TOS, further details of the stack structure are not shown.

```
MCS-96 MACRO ASSEMBLER   SQRT                                          05/12/86 10:44:30 PAGE   1
DOS MCS-96 MACRO ASSEMBLER, V1.1
SOURCE FILE: ROOT2.A96
OBJECT FILE: ROOT2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB
ERR LOC OBJECT        LINE       SOURCE STATEMENT
                        1  ;
                        2  sqrt module
                        3  ;
                        4  ; 32 bit integer square root for the 8096
                        5  ;
                        6  public qstk_isqrt            ;      TOP←  SQUARE_ROOT(TOP)
                        7  extrn interr:entry           ; Integer error routine
                        8  ;
                        9  ;   id stags for stack integer routines
     0019               10  isqrt_id    equ     19h
                        11  ;
                        12  ; error codes
                        13  ;
     0000               14  overflow        equ     00h
     0001               15  paramerr        equ     01h
     0002               16  invalid_input   equ     02h
                        17
     001C               18      oseg at 1ch
                        19  ;   ===========
     001C               20  ax:     dsw 1
       001C             21   al equ ax:byte
       001D             22   ah equ (ax+1):byte
     001E               23  dx:     dsw 1
     0020               24  cx:     dsw 1
     0022               25  bx:     dsw 1
       0018             26  sp      equ 18h:word
                        27
                        28
     0030               29      oseg at 30h
                        30  ;   ===========
     0030               31  qstk_reg:
     0030               32      dsl   1                 ; make sure of alignment
       0030             33  next    equ qstk_reg:word   ; pointer to next element in the arg stack.
       0032             34  tos_tag equ (qstk_reg+2):word
     0034               35  tos_value:
     0034               36      dsl   1                 ; 32 bit integer
                        37  ;
     0000               38      cseg
                        39  ;   ====
                        40  bl  macro param
                        41      bnc   param
                        42      endm
                        43
                        44  bhe macro param
                        45      bc    param
                        46      endm
                        47  $eject
```

```
MCS-96 MACRO ASSEMBLER  SQRT                                          05/12/86 10:44:30 PAGE   2
ERR LOC  OBJECT           LINE       SOURCE STATEMENT
    0000                    48       cseg
                            49   ;   ====
                            50   ;
    0000                    51   qstk_isqrt:
                            52   ; Takes the square root of the long integer in TOS
                            53   ; TOS--> Top of argument stack
                            54   ;   iTOS - iSQRT(TOS)
                            55   ;
      0020                  56       Xold set cx
    0000 A0341C             57       ld      ax,tos_value
    0003 A0361E             58       ld      dx,(tos_value+2)
    0006 371F07             59       bbc     (dx+1),7,qsi05      ; if (TOS < 0)
    0009 C90119             60       push    #(isqrt_id*256+paramerr)
    000C EF0000         E   61       call    interr              ;    Call interr.
    000F F0                 62       ret                         ;    Exit
    0010                    63   qsi05:
    0010 0F221C             64       normal  ax, bx
    0013 DF3B               65       be      qstk_isqrt0
    0015 991022             66       cmpb    bx,#16              ; if (TOS < 2**16)
    0018 DA06               67       ble     qsil0
    001A A1FF0020           68       ld      Xold, #0ffh         ;    Use 0ffh as first estimate.
    001E 200A               69       br      qstk_isqrtloop
    0020                    70   qsil0:
    0020 180122             71       shrb    bx,#1               ; else
    0023 A1FFFF20           72       ld      Xold, #0ffffh       ;    Base the first estimate on the
    0027 082220             73       shr     Xold, bx            ;    number of leading zeroes in TOS.
    002A                    74   qstk_isqrtloop;
    002A A0341C             75       ld      ax,tos_value        ; do
    002D A0361E             76       ld      dx,(tos_value+2)    ;    if (The divide will overflow)
    0030 88201E             77       cmp     dx,Xold             ;       The loop is done.
                            78       bhe     qstk_isqrt_done
    0035 8C201C             80       divu    ax,Xold             ;    if ( (ax=TOS/Xold) >= Xold)
    0038 88201C             81       cmp     ax,Xold             ;       The loop is done.
                            82       bhe     qstk_isqrt_done
    003D 0122               84       clr     bx                  ;    Xold=(ax+Xold)/2
    003F 641C20             85       add     Xold,ax
    0042 A40022             86       addc    bx,0
    0045 0C0120             87       shrl    Xold,#1
    0048 27E0               88       br      qstk_isqrtloop      ; while (The loop is not done)
    004A                    89   qstk_isqrt_done:
    004A A02034             90       ld      tos_value,Xold      ; TOS=00:Xold
    004D A00036             91       ld      (tos_value+2),0
    0050                    92   qstk_isqrt0:
    0050 F0                 93       ret                         ; Exit
    0051                    94       end

ASSEMBLY COMPLETED.  NO ERROR(S) FOUND.
```