ISqrt.bas

An Integer square root is the nearest whole number smaller than the full square root answer. So the integer square root of 10 is 3 instead of 3.162277. You'd get the same answer as `INT(SQR(x))` with an integer square root function.

For things like games programming, this is often near enough - for example, the distance formula, based on Pythagoras' equation $A^2 = B^2 + C^2$ only works if you square root the answer. If you need to find the distances between your items, then you're going to be doing a lot of square roots, and you're going to need to do them FAST (that said the even faster solution might be this one: distance.bas or if you don't need the actual distance just the answer to the question *"which one is further away?"* then not square rooting is needed, and comparing distance1^2 with distance2^2 still tells you which is nearer. Berksman written in ZX Basic, does this trick of never doing the square root part, for example.

Anyway, this function returns integer square roots. For numbers less than 65536, it's about 100 times faster, because it can do 16 bit calculation. For longer numbers, it has to do 32 bit calculations, which are less than optimal on an 8 bit processor! It's still about 50 times faster than the ROM routine, however.

If you want completely accurate results, you should use the floating point fast routine over at fSqrt.bas.

```
FUNCTION FASTCALL iSqrt (num as uLong) as uInteger
REM incoming is DEHL
REM output is HL

asm
    LD A,D
    OR E
    JP Z, sqrtLF16bit ; we're inside a 16 bit number. We can use the faster version.

    LD b,16 ; b times round
    EXX ; Out to root and rem - we're doing most of this in alternate registers.
    LD DE,0
    LD HL,0 ; DEHL = remainder
    LD BC,0 ; BC = root
    EXX   ;back to num and loop
sqrtLFasmloop:
    EXX  ; out to root and rem

    SLA  C ; root <<= 1
    RL  B   ;

    SLA L ; rem=rem<<1
    RL  H  ;
    RL  E    ;
    RL  D     ;

    SLA L ; rem=rem<<1
    RL  H  ;
    RL  E    ;
    RL  D     ;
    EXX  ; back to Num and loop

    LD a,d     ; A = inputnum>>30
    AND 192
    RLCA
    RLCA

    SLA  L ; num <<= 1
    RL  H
    RL  E
    RL  D

    SLA  L ; num <<= 1
    RL  H
    RL  E
    RL  D

    EXX  ; out to root and rem

    ADD A,L     ; a=a+L              ; REM=REM+num>>30
    LD L,A      ; a-> L               ;
    JR NC, sqrtLFasmloophop1           ;
    INC H
    JR NC, sqrtLFasmloophop1
    INC DE      ;

sqrtLFasmloophop1:
    INC BC                      ; root=root+1

sqrtLFasmloophop2:
    ; DEHL = Remainder
    ; BC = root

    ; if rem >= root then
    LD A,D
    OR E
    JR NZ, sqrtLFasmthen ; if rem > 65535 then rem is definitely > root and we go to true

    LD A, H
    CP B
    JR C, sqrtLFasmelse ; H<B - that is rem<root so rem>=root is false and we go to else
    JR NZ, sqrtLFasmthen ; H isn't zero though, so we could do a carry from it, so we're good to say
    ; if h is out, then it's down to L and C
    LD A,L
    CP C
    JR C, sqrtLFasmelse ; L<C - that is rem<root so rem>=root is false and we go to else
    ; must be true - go to true.

sqrtLFasmthen:
    ;remainder=remainder-root
    AND A ; clear carry flag
    SBC HL,BC ; take root away from the lower half of rem.
    JP NC, sqrtLFasmhop3 ; we didn't take away too much, so we're okay to loop round.

    ; if we're here, we did take away too much. We need to borrow from DE
    DEC DE ; borrow off DE

sqrtLFasmhop3:
    INC BC ;root=root+1
    JP sqrtLFasmloopend

    ;else
sqrtLFasmelse:
    DEC BC ;root=root-1
    ;end if


sqrtLFasmloopend:
    EXX  ; back to num
    DJNZ sqrtLFasmloop

    EXX ; out to root and rem
    PUSH BC

    EXX ; back to normal
    POP HL
    SRA  H
    RES 7,H
    RR  L       ; Hl=HL/2 - root/2 is the answer.
    jr sqrtLFexitFunction

sqrtLF16bit:

    ld  a,l
    ld  l,h
    ld   de,0040h   ; 40h appends "01" to D
    ld   h,d
    ld b,7

sqrtLFsqrt16loop:
    sbc   hl,de      ; IF speed is critical, and you don't mind spending the extra bytes,
                   ; you could unroll this loop 7 times instead of DJNZ.

    ; deprecated because of issues - jr nc,$+3 (note that if you unroll this loop, you'll need 7 labe
    jr    nc,sqrtLFsqrthop1
    add   hl,de


sqrtLFsqrthop1:
    ccf
    rl    d
    rla
    adc   hl,hl
    rla
    adc   hl,hl

    DJNZ sqrtLFsqrt16loop

    sbc   hl,de      ; optimised last iteration
    ccf
    rl   d
    ld h,0
    ld l,d
    ld de,0
sqrtLFexitFunction:
    end asm
END FUNCTION
```