Performance of Arithmetic Shifting Vs. IMUL and IDIV i486 Instructions

Isaac G. Styles

12/1/15

East Tennessee State University

Integer division or multiplication by a power of 2 can be accomplished by bit shifting instructions. The goal was to determine if the AMD Phenom II, codename "Deneb", used internal optimizations within the i486 IDIV and IMUL assembly instructions to increase the performance of integer division and multiplication.

Implementation

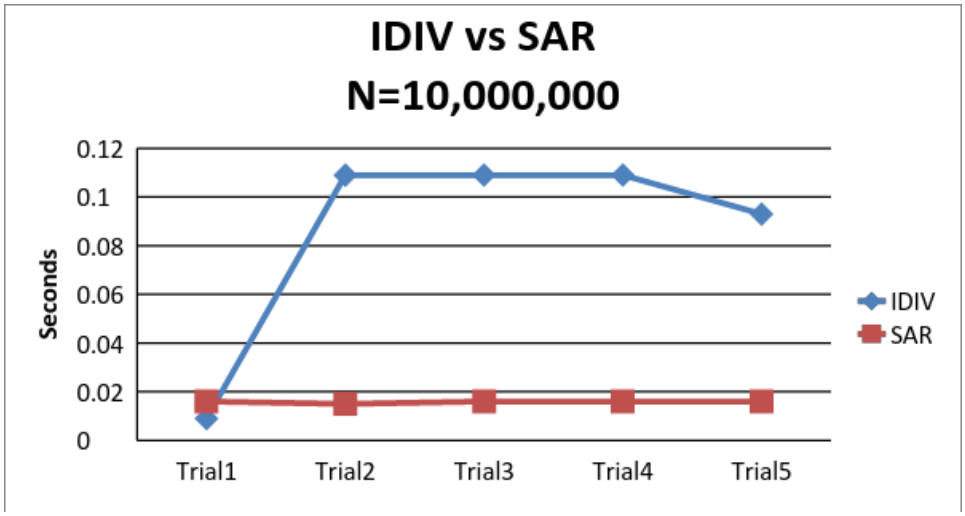
Shifting was performed on the 32-bit EAX register. The divisor and multiplier were set to the 16-bit value 2d to ensure the attributes and results didn't exceed 32-bit. The multiplicand and dividend were set to 500d prior to each arithmetic operation. The loop was performed 10,000,000 times, and the start and stop times were recorded. Five trials were performed. The process priority was set to High to minimize the number of interrupts.

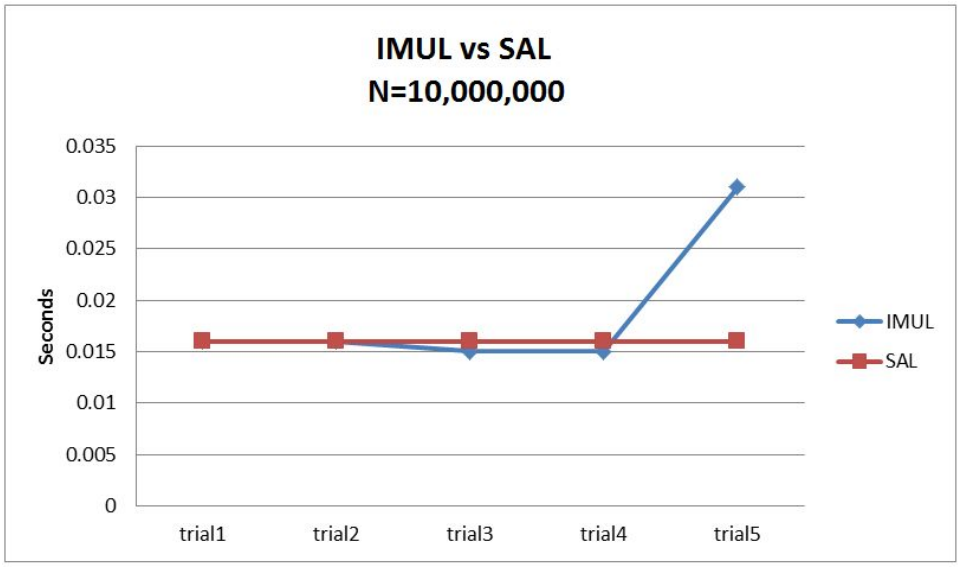
Results

When using IDIV and IMUL, the best case scenario took the equivalent time as their respective shifting instruction. This suggests that the AMD "Deneb" core does use bit shifting within these arithmetic instructions. However, the variability of the time required for the arithmetic instructions was significantly higher. The Variance ($\sigma 2$) for IDIV was $1.51 \times 10^{-3} sec^2$ while the variance for SAR was $1.59 \times 10^{-7} sec^2$, which is negligible. Similarly the variance for IMUL was $4.24 \times 10^{-5} sec^2$ while the variance for SAL was 0 sec^2. Although the best case for all four instructions was the same, the average run-time for shifting instructions was lower. The average for IDIV was 0.0858 sec and the average for SAR was 0.0158 sec. The average for IMUL was 0.0188 sec and the average for SAL was 0.016 sec.

Summary

The IDIV and IMUL operations are capable of performing as fast as SAR and SAL, however their worst case run-time is significantly higher. Shifting instructions are more consistent when multiplying or dividing integers by a power of 2.





Elapsed Time of .486 Assembly Instructions (sec)					
	N=10,000,000				
	Trial1	Trial2	Trial3	Trial4	Trial5
IDIV	0.009	0.109	0.109	0.109	0.093
SAR	0.016	0.015	0.016	0.016	0.016
IMUL	0.016	0.016	0.015	0.015	0.032
SAL	0.016	0.016	0.016	0.016	0.016

Source code: Assembled and Linked with MASM 5.10

```
.486                                    ;Generates the Assembly instruction set
.model flat                             ;Generates the 32-bit code but without using
.stack 100h                             ;allocates 100h bytes to the stack
    ExitProcess PROTO Near32 stdcall, dVal:dword
    getch       PROTO Near32 stdcall
    putstring   PROTO Near32 stdcall, lpStringToPrint:dword
    putch       PROTO Near32 stdcall, bVal:byte
    getTime     PROTO Near32 stdcall, lpStringToHold:dword
    pressEnterToContinue PROTO Near32 stdcall, lpStringToPrint:dword
    newline macro
        INVOKE putch, 10
        INVOKE putch, 13
    endm
.data
    loopThrough dword 10000000d                     ;times to repeat loops
    enterToStart byte 10,13,9,"Press ENTER to start",0
    t1 byte 40 dup(?)
    t2 byte 40 dup(?)
    t3 byte 40 dup(?)
    t4 byte 40 dup(?)
.code
_start:
    INVOKE pressEnterToContinue, addr enterToStart
    newline
    mov bx, 2d                                      ;bx is divisor and multiplicand
    mov ecx, loopThrough                            ;repeat idiv N times

    push ecx
    INVOKE getTime, addr t1                         ;record start time for idiv / imul
    pop ecx
divloop:
    mov eax, 500d
    CWD             ;dx:ax
    idiv bx
    loop divloop

    INVOKE getTime, addr t2                         ;record end time for idiv / imul

    mov ecx, loopThrough                            ;repeat shift N times

    push ecx
    INVOKE getTime, addr t3                         ;record start time for shift operations
    pop ecx
shiftLoop:
    mov eax, 500d
    CWD             ;dx:ax
    SAR eax, 1
    loop shiftLoop
    INVOKE getTime, addr t4                         ;record end time for shift operations

    INVOKE putstring, addr t1
    newline
    INVOKE putstring, addr t2
    newline
    newline
    INVOKE putstring, addr t3
    newline
    INVOKE putstring, addr t4
```