```
;Author information
;  Author name: Art Grichine
;  Author email: ArtGrichine@gmail.com
;Course information
;  Course number: CPSC240
;  Assignment number: 3
;  Due date: 2014-Feb-25
;Project information
;  Project title: The Powers Of e (Assignment 3)
;  Purpose: Preform Taylor Series (e^x) calculations on 4 user inputs to a user defined amount of iterations, output initial user
;           'x' values, result of Taylor Series for each given e^x solution, and elapsed time of the calculations from the system
;           clock in tics, nanoseconds, and seconds.
;  Status: No known errors
;  Project files: ThePowersOfe.cpp, ThePowersOfe.asm, debug.inc, debug.asm
;Module information
;  This module's call name: ThePowersOfe
;  Language: X86-64
;  Syntax: Intel
;  Date last modified: 2014-Feb-20
;  Purpose: Intake four values from user, intake a Taylor Series iteration, preform e^x calculation to the user defined amount of
;           iterations. Output initial user 'x' values, result from e^x and the elapsed time of calculations in tics, ns & seconds
;  File name: ThePowersOfe.asm
;  Status: In production.  No known errors.
;  Future enhancements: None planned
;Translator information
;  Assemble: nasm -f elf64 -l ThePowersOfe.lis -o ThePowersOfe.o ThePowersOfe.asm
;References and credits
;  CSUF/Professor Floyd Holliday: http://holliday.ecs.fullerton.edu
;Print information
;  Page width: 132 columns
;  Begin comments: 65
;  Optimal print specification: Landscape, 9 points or smaller, monospace, 8Â1⁄2x11 paper
;
;===== Begin code area ==================================================================================================

%include "debug.inc"                                            ;This file contains the subprogram to be tested with this test program.

extern printf                                                   ;External C++ function for writing to standard output device

extern scanf                                                    ;External C++ function for obtaining user input

global ThePowersOfe                                             ;This makes ThePowersOfe callable by functions outside of this file.

segment .data                                                   ;Place initialized data here

;===== Message Declarations =============================================================================================

welcome db 10, "Welcome to e^x programmed by Art Grichine!", 10,
        db     "This program was tested on a MacBook Pro (late 2013) running Haswell i7 at 2.6GHz.", 10,
        db     "The program will compute e raised to any value x with a high level of accuracy.", 10, 0
```

```
enter_number_x db  10, "Enter 4 floating point numbers for x separated by white space and press enter: ", 0

enter_number_taylor_series db "Enter the number of terms in the Taylor series: ", 0

thank_you db "Thank you.", 10, 0

results db 10, "The computed results are as follows:", 10
        db     "           e^%3.5lf = %5.20lf", 10
        db     "           e^%3.5lf = %5.20lf", 10
        db     "           e^%3.5lf = %5.20lf", 10
        db     "           e^%3.5lf = %5.20lf", 10
        db     "Enjoy your exponents. The last of your values will be returned to the driver.", 10, 0

system_clock_pre db 10, "The system clock is %10ld, and parallel computations have begun.", 10, 0

system_clock_post db 10, "The computations have completed and the system clock was %10ld at completion.", 10, 0

elapsed_time db 10, "The elapsed time was %7ld tics = %7.1lf ns ≈ %1.8lf seconds.", 10, 0

stringformat db "%s", 0                                         ;general string format

formatFourFloats db "%lf %lf %lf %lf", 0                        ;this format will absorb four 'x' user inputs of float type 64-bit

formatOneFloat   db "%lf", 0                                    ;this format will absorb Taylor Series input of float type 64-bit

segment .bss                                                    ;Place un-initialized data here.

        ;This segment is empty

segment .text                                                   ;Place executable instructions in this segment.

ThePowersOfe:                                                   ;Entry point.  Execution begins here.

;=========== Back up all the integer registers used in this program ================================================================

push rbp                                                        ;Backup the stack base pointer
push rdi                                                        ;Backup the destination index
push rsi                                                        ;Backup the source index

;============  Preliminary =========================================================================================================

vzeroall                                                        ;zeros out all SSE registers

;=========== Initialize divider register ===========================================================================================
                                                                ;for our calculations we must set 4 SSE registry's to 1.0 to
                                                                ;accomidate for our algorithm: count, nth term, accum
mov  rax, 0x3ff0000000000000                                    ;copy HEX 1.0 value onto rax
push rax                                                        ;push rax value onto the stack for broadcast operation
vbroadcastsd ymm10, [rsp]                                       ;makes ymm10 all 1.0, this will be our count
```

```
vbroadcastsd ymm9,  [rsp]                                          ;makes ymm9 all 1.0, this will hold our nth term
vbroadcastsd ymm8,  [rsp]                                          ;makes ymm8 all 1.0, this will be our accumulator
vbroadcastsd ymm14, [rsp]                                          ;ymm14 we will use ymm14 to incriment count by 1.0 each iteration
pop  rax                                                           ;push operand must be followed by a pop operation when complete

;=========== Show the initial message ==================================================================================

xor  rax, rax                                                      ;tell printf not to expect any doubles in upcoming call
mov  rdi, stringformat                                             ;simple format indicating string ' "%s",0 '
mov  rsi, welcome                                                  ;display: Welcome Message, Name, Machine, Purpose of Assignment
call printf                                                        ;display welcome message from the driver program (.cpp)

;============ Input for X values =======================================================================================

;==== Display message for x ====
xor  rax, rax                                                      ;satisfies printf function, expects no doubles in upcoming printf
mov  rdi, stringformat                                             ;"%s"
mov  rsi, enter_number_x                                           ;asks user to enter four FPU numbers seperated by a space
call printf                                                        ;display user prompt for 'x' values

push qword 0                                                       ;allocate storage for an input number on int stack
push qword 0                                                       ;allocate storage for an input number on int stack
push qword 0                                                       ;allocate storage for an input number on int stack
push qword 0                                                       ;allocate storage for an input number on int stack

;==== Grab data for x ====
xor  rax, rax                                                      ;clear rax registry
mov  rdi, formatFourFloats                                         ;formats input of scanf to recieve four numbers "%lf %lf %lf %lf"
mov  rsi, rsp                                                      ;assign register to copy stack pointer location to absorb x1
mov  rdx, rsp                                                      ;assign rdx to stack pointer in preperation for x2
add  rdx, 8                                                        ;assign register to copy stack pointer location to absorb x2
mov  rcx, rsp                                                      ;assign rcx to stack pointer in preperation for x3
add  rcx, 16                                                       ;assign register to copy stack pointer location to absorb x3
mov  r8, rsp                                                       ;assign r8 to stack pointer in preperation for x4
add  r8, 24                                                        ;assign register to copy stack pointer location to absorb x4
call scanf                                                         ;scan all four user inputs for 'x' and place in appropriate space

;============ Move int stack into ymm registers ========================================================================

vmovupd ymm11, [rsp]                                               ;place user x values onto ymm11 for upcoming manipulation
pop rax                                                            ;deallocate memory allocation for input
pop rax                                                            ;deallocate memory allocation for input
pop rax                                                            ;deallocate memory allocation for input
pop rax                                                            ;deallocate memory allocation for input

;============ Input Taylor Series value ================================================================================

;==== Display message for Taylor Series ====
xor  rax, rax                                                      ;satisfies printf function, expects no doubles in upcoming printf
mov  rdi, stringformat                                             ;"%s"
```

```
mov  rsi, enter_number_taylor_series                          ;Format: "Enter the number of terms in the Taylor series: "
call printf                                                   ;ask the user to enter the Taylor Series iteration

push qword 0                                                  ;allocate storage for a user input number on stack

;==== Grab data for Taylor Series ====
xor  rax, rax                                                 ;clear rax register, required for scanf
mov  rdi, formatOneFloat                                      ;format to absorb one number: "%lf"
mov  rsi, rsp                                                 ;set location of Taylor Series
call scanf                                                    ;scan users input into the registry using the .cpp driver

;==== Place Taylor Series number on AVX ====

vmovupd ymm15, [rsp]                                          ;place Taylor Series number on xmm15 to compare for iteration
pop rax                                                       ;deallocate memory allocation for input

;==== Say "Thank You" ====
xor  rax, rax                                                 ;tell printf not to look for any doubles in SSE
mov  rdi, stringformat                                        ;"%s"
mov  rsi, thank_you                                           ;print 'Thank you.'
call printf                                                   ;Say 'Thank you.' on the command terminal

;============ Read system clock pre-calculations===================================================================
                                                              ;System clock-speed found in 1/2 of rdx and 1/2 of rax, to get the
                                                              ;accurate clock-speed (in tics) we must read the clock and then
                                                              ;combine the two registers (rdx:rax) into one for the full reading.
;==== get system clock ====
cpuid                                                         ;stop 'look ahead' for cpu; lets us get a more accurate clock speed
rdtsc                                                         ;read cpu clock, values stored in rdx:rax
mov qword r13, 0                                              ;zero out r13 register for upcoming calculations, will hold rax
mov qword r12, 0                                              ;zero out r12 register for upcoming calculations, will hold rdx
mov       r13, rax                                            ;place rdx value onto r13 in preparation to combine rax/rdx
mov       r12, rdx                                            ;place rdx value onto r12 in preparation to combine rax/rdx
shl       r12, 32                                             ;shift r12 32 bits
or        r12, r13                                            ;combine r12 and r13, r12 has entire system clock count now

;==== output system clock ====
mov rsi, r12                                                  ;place timer value from rax into rsi in preperation for output
xor rax, rax                                                  ;zero-out rax register
mov rdi, system_clock_pre                                     ;assign output format to rdi for printf output
call printf                                                   ;print system clock tics pre-calculation

;============ Calculate e^x =======================================================================================

iterate_again:                                                ;iterate through until value is =to xmm15 (our taylor series input)

vmulpd ymm9, ymm11                                            ;x^n
vdivpd ymm9, ymm10                                            ;(x^n)/n!
vaddpd ymm8, ymm9                                             ;1 + (x^n)/n! + (x^n+1)/(n+1)! + etc...
```

```
ucomisd xmm10, xmm15                                        ;compare count (xmm7) with our users Taylor Series number (xmm15)
vaddpd  ymm10, ymm14                                        ;increment count by 1
jb iterate_again                                            ;if xmm7 < xmm15 jump to 'iterate_again:' jb is 'jumb below', if
                                                            ;we wanted to compare xmm7 > 15 we would use ja 'jumb above'
;============ Read system clock post-calculations==================================================================================

;==== get system clock ====
cpuid                                                       ;stop 'look ahead' for cpu
rdtsc                                                       ;read cpu clock, values stored in rdx:rax
mov qword r15, 0                                            ;zero out r15 register for upcoming calculations, will hold rax
mov qword r14, 0                                            ;zero out r14 register for upcoming calculations, will hold rdx
mov       r15, rax                                          ;place rax value onto r15 in preparation to combine rax/rdx
mov       r14, rdx                                          ;place rdx value onto r14 in preparation to combine rax/rdx
shl       r14, 32                                           ;shift r14 32 bits
or        r14, r15                                          ;combine r14 and r15, r14 has entire system clock count now

;==== output system clock ====
mov rsi, r14                                                ;place timer value from rax into rsi in preperation for output
xor rax, rax                                                ;zero-out rax register
mov rdi, system_clock_post                                  ;assign output format to rdi for printf output
call printf                                                 ;print system clock tics post-calculation

;========== Elapsed time ==========================================================================================================
                                                            ;our values of clockspeed pre-calculations is in r12, clockspeed
                                                            ;post-calculations is in r14
;==== calculate elapsed time (tics) ====
sub r14, r12                                                ;subtract the clockspeed pre-calculation from post-calc clockspeed
                                                            ;this gives us our elapsed time

                                                            ;bc we are doing floating point calculations it is easier to place
                                                            ;values onto the SSE registry and do the calculations there.

;==== calculate elapsed time (ns) ====                      ;our ns conversion formula is (clockspeed (in tics) * 10) / 26 = ns
mov       r11, 10                                           ;need to increase numerator by factor of 10, set up upcoming calc
cvtsi2sd xmm4, r11                                          ;convert 10 into floating point and place onto xmm4
cvtsi2sd xmm5, r14                                          ;convert our clockspeed (in tics) and place onto xmm5
vmulpd   ymm4, ymm5                                         ;multiply clockspeed (in tics) by factor of 10
mov       r11, 26                                           ;int 26 onto r11 so that we may divide our value by(machine spd*10)
cvtsi2sd xmm5, r11                                          ;convert 26 into floating point and place onto xmm5
vdivpd   ymm4, ymm5                                         ;divide (tics * 10) by known cpu speed (2.6Ghz * 10) to get real ns
vmovupd  ymm0, ymm4                                         ;output for nanoseconds ready on ymm0

;==== calculate elapsed time (sec) ====                     ;ns --> sec = ns/1billion
mov r11, 1000000000                                         ;place 1billion onto r11 to prepare for floatpnt conversion on SSE
cvtsi2sd xmm5, r11                                          ;converst 1billion into float number and place on xmm5
vdivpd ymm4, ymm5                                           ;divide ns/1billion = elapsed time in seconds
vmovupd ymm1, ymm4                                          ;place elapsed time in seconds onto ymm1 for output

;==== output elapsed time ====                              ;elapsed time in tics sits on r14, ns on ymm0, sec on ymm1
mov rsi, r14                                                ;place elapsed time in tics on to rsi for output
```

```
mov rax, 2                                          ;tell printf to find the first two numbers(ns/sec)on SSE(xmm0,xmm1)
mov rdi, elapsed_time                               ;format output for elapsed time in tics, ns, and seconds
call printf                                         ;output elapsed time in tics, ns, and seconds


;========== Output e^x =============================================================================================
                                                    ;Output is formated for one big print of 8 values, 4 user initial
                                                    ;'x' values and 4 e^x totals. Initial 'x' values are found on ymm11
                                                    ;and e^x totals found on ymm8. SSE and AVE registers must be moved
                                                    ;around to the low xmm0 - xmm7 slots. This format must look like
                                                    ;this before print: (xmm0)x1,e^x1,x2,e^x2,x3,e^x3,x4,(xmm7)e^x4

;==== prepare x output ====                         ;output data for users initial 'x' values is sitting in ymm11
vmovupd   ymm0, ymm11                               ;move into ymm0 for x1
vpermilpd ymm2, ymm11, 0x1B                         ;swap, places x2 value in ymm2 for output
vextractf128 xmm4, ymm11, 1                         ;shift of ymm into xmm, ABCC --> CCAB, places x3 in xmm4 for output
vpermilpd ymm6, ymm4, 0x1B                          ;swap, places x4 value in ymm6 for output

;==== prepare e^x value output ====                 ;output data for e^x totals sitting in ymm8
vmovupd   ymm1, ymm8                                ;move into ymm1 for e^x1
vpermilpd ymm3, ymm8, 0x1B                          ;swap, places e^x2 value in ymm3 for output
vextractf128 xmm5, ymm8, 1                          ;shift of ymm into xmm, ABCC --> CCAB, places e^x3 in xmm5 for out
vpermilpd ymm7, ymm5, 0x1B                          ;swap, places e^x2 value in ymm3 for output

;==== print ====
mov  rax, 8                                         ;tell printf to expect 8 values on the upcoming print
mov  rdi, results                                   ;prepare format for our 8 values on printf
call printf                                         ;print x and e^x values to the terminal

;=========== Now cleanup and return to the caller ==================================================================

pop rsi                                             ;Restore original value
pop rdi                                             ;Restore original value
pop rbp                                             ;Restore original value

mov qword rax, 1                                    ;Return value of 1 to the driver

ret                                                 ;Return to driver program (.cpp)

;========== End of program ThePowersOfe.asm=========================================================================
```