

comment ~*****

Name: Bubba
SSN: 123-45-7890
Course: COSC 2410
Instructor: Dr. Mirkovic
Section: 09152

Program: hw1

Description:

This takes input from the keyboard and computes the weighted average based on the following formula

$$\text{avg} = \frac{(\text{weighta} * a) + (\text{weightb} * b)}{(\text{weighta} + \text{weightb})}$$

The average and the remainder of the division are displayed to screen.

In part 2 of the program, the amount of cycles it takes the processor to execute these calculations is computed. To avoid undue delays, I/O is disabled and the variables are assigned as follows

a=0
b=0
weighta=1
weightb=1

To compute the number of cycles the "CPUID" instruction is called to force serialization of the commands and then "RDTSC" is executed to retrieve the current cycle count. After the calculations are preformed, the cycle count is retrieved and the previously obtained cycle count is subtracted from the current one to obtain the number of cycles to complete the given calculations. This is repeated in "chunks" of 1, 2, and 4 times. This result is displayed to the screen after each "chunk." To optimize the execution time, the processor is "warmed up" by completing successive calls to "CPUID," and "RDTSC" while saving the "RDTSC" output over and over. This assures that the variable used to store the cycle time is in the processor cache and the calculations will not suffer the look up time of waiting for main memory

Input:

a - the "a" value to be used in computing the weighted average
b - the "b" value to be used in computing the weighted average
weighta - the weight to be multiply variable "a" by
weightb - the weight to be multiply variable "b" by

Output:

None

Return:

None

```

~*****
.model small
.586
.stack 100h
.nolist
;*****
;Includes
;*****
include bios.inc

.list
.data
;*****
;Defintion declarations
;*****
CR equ 0Dh
LF equ 0Ah

;*****
;Variables and constant declarations
;*****
mess0 db CR,LF,CR,LF,"Iterations to go:"
mess1 db CR,LF,"Enter value for a:"
mess2 db "Enter value for the weight of a:"
mess3 db "Enter value for b:"
mess4 db "Enter value for the weight of b:"
mess5 db CR,LF,"The resultant weighted average:"
mess6 db CR,LF,"The remainder weighted average:"
mess7 db CR,LF,"The number of cycles:"
mess0len equ lengthof mess0
mess1len equ lengthof mess1
mess2len equ lengthof mess2
mess3len equ lengthof mess3
mess4len equ lengthof mess4
mess5len equ lengthof mess5
mess6len equ lengthof mess6
mess7len equ lengthof mess7

;signed integer variables
;
weighta sword 0
weightb sword 0
a sword 0
b sword 0
result sdword 0
subtime dd 0

.code
;*****
;Function prototypes
;*****
extern GetDec:near, PutDec:near
displaymsg proto near c m:word,l:byte

;*****

```

```
;Start of code
;*****
```

```
main proc
```

```
.startup
;*****
;Part 1
;*****
```

```
@CIs ;clear screen
@Setcsrpos 0,0 ;move cursor to row 0, column 0
```

```
mov eax, 0 ;clear out eax register
mov ebx, 0 ;clear out ebx register
mov ecx, 0 ;clear out ecx register
mov edx, 0 ;clear out edx register
mov cx, 5 ;initialize the loop variable
```

question:

```
label question
mov ax, cx
invoke displaymsg, addr mess0, mess0len ;display message 0
call PutDec
```

```
pushad
invoke displaymsg, addr mess1, mess1len ;display message 1
call GetDec ;read an integer from stdin
mov a, ax ;move the integer to var a
```

```
invoke displaymsg, addr mess2, mess2len ;display message 2
call GetDec ;read an integer from stdin
mov weighta, ax ;move the integer to var weighta
imul a ;signed multiply ax by var a
mov SWORD PTR result, ax ;place result of ax into var result
```

```
invoke displaymsg, addr mess3, mess3len ;display message 3
call GetDec ;read an integer from stdin
mov b, ax ;move the integer into var b
invoke displaymsg, addr mess4, mess4len ;display message 4
call GetDec ;read an integer from stdin
```

```
mov weightb, ax ;move the integer into var weightb
imul b ;signed multiply ax by var b
add ax, SWORD PTR result ;add the var result to register ax
```

```
mov bx, weighta ;move the var weight a into register bx
add bx, weightb ;add var weightb to register bx
```

```
idiv bx ;mulitply ax by bx
```

```
invoke displaymsg, addr mess5, mess5len ;display message 5
call PutDec ;display the quotient of the division to stdout
invoke displaymsg, addr mess6, mess6len ;display message 6
mov dx, ax ;move dx to ax
call PutDec ;display the remainder of the division to stdout
popad
```

```
sub cx, 1 ;decrement CX to reflect iteration number
```

```

cmp cx, 0                ;compare CX to zero
jne question            ;if CX does not equal zero then jump to label question

;*****
;Part 2 - warm up (do this at least three times)
;*****
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
sub eax, subtime        ;subtract the stored from the current result

;*****
;Part 2 - Single run
;*****
cpuid                   ;force serialization of instructions
rdtsc                   ;get the cycles
mov subtime, eax        ;move the cycles out to memory

```

```

mov a, 0 ;move the integer to var a
mov weighta, 1 ;move the integer to var weighta
mov ax, weighta ;move the variable weighta into ax
imul a ;signed multiply ax by var a
mov SWORD PTR result, ax ;place result of ax into var result
mov b, 0 ;move the integer into var b
mov weightb, 1 ;move the integer into var weightb
mov ax, weightb
imul b ;signed multiply ax by var b
add ax, SWORD PTR result ;add the var result to register ax
mov bx, weighta ;move the var weight a into register bx
add bx, weightb ;add var weightb to register bx
idiv bx ;mulitply ax by bx

cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result
invoke displaymsg, addr mess7, mess7len ;display message 7
call PutDec ;display the quotient of the division to stdout

;*****
;Part 2 - Double warm up
;*****
cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles

```

```

mov subtime, eax           ;move the cycles out to memory
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
sub eax, subtime          ;subtract the stored from the current result

cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
mov subtime, eax          ;move the cycles out to memory
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
sub eax, subtime          ;subtract the stored from the current result

cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
mov subtime, eax          ;move the cycles out to memory
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
sub eax, subtime          ;subtract the stored from the current result

cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
mov subtime, eax          ;move the cycles out to memory
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
sub eax, subtime          ;subtract the stored from the current result

cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
mov subtime, eax          ;move the cycles out to memory
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
sub eax, subtime          ;subtract the stored from the current result

;*****
;Part 2 - Double run
;*****
cpuid                     ;force serialization of instructions
rdtsc                     ;get the cycles
mov subtime, eax          ;move the cycles out to memory

mov a, 0                  ;move the integer to var a
mov weighta, 1            ;move the integer to var weighta
mov ax, weighta           ;move the variable weighta into ax
imul a                    ;signed multiply ax by var a
mov SWORD PTR result, ax ;place result of ax into var result
mov b, 0                  ;move the integer into var b
mov weightb, 1            ;move the integer into var weightb
mov ax, weightb           ;move the integer into var weightb
imul b                    ;signed multiply ax by var b
add ax, SWORD PTR result ;add the var result to register ax
mov bx, weighta           ;move the var weight a into register bx
add bx, weightb           ;add var weightb to register bx
idiv bx                   ;mulitply ax by bx

```

```

mov a, 0 ;move the integer to var a
mov weighta, 1 ;move the integer to var weighta
mov ax, weighta ;move the variable weighta into ax
imul a ;signed multiply ax by var a
mov SWORD PTR result, ax ;place result of ax into var result
mov b, 0 ;move the integer into var b
mov weightb, 1 ;move the integer into var weightb
mov ax, weightb
imul b ;signed multiply ax by var b
add ax, SWORD PTR result ;add the var result to register ax
mov bx, weighta ;move the var weight a into register bx
add bx, weightb ;add var weightb to register bx
idiv bx ;mulitply ax by bx

cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result
invoke displaymesg, addr mess7, mess7len ;display message 7
call PutDec ;display the quotient of the division to stdout

;*****
;Part 2 - Quad warm up
;*****
cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory

```

| | |
|------------------|--|
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| mov subtime, eax | ;move the cycles out to memory |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| mov subtime, eax | ;move the cycles out to memory |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| mov subtime, eax | ;move the cycles out to memory |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| mov subtime, eax | ;move the cycles out to memory |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| mov subtime, eax | ;move the cycles out to memory |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| cpuid | ;force serialization of instructions |

```

rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory
cpuid ;force serialization of instructions
rdtsc ;get the cycles
sub eax, subtime ;subtract the stored from the current result

;*****
;Part 2 - Quad run
;*****
cpuid ;force serialization of instructions
rdtsc ;get the cycles
mov subtime, eax ;move the cycles out to memory

mov a, 0 ;move the integer to var a
mov weighta, 1 ;move the integer to var weighta
mov ax, weighta ;move the variable weighta into ax
imul a ;signed multiply ax by var a
mov SWORD PTR result, ax ;place result of ax into var result
mov b, 0 ;move the integer into var b

```

| | |
|---|--|
| mov weightb, 1 | ;move the integer into var weightb |
| mov ax, weightb | |
| imul b | ;signed multiply ax by var b |
| add ax, SWORD PTR result | ;add the var result to register ax |
| mov bx, weighta | ;move the var weight a into register bx |
| add bx, weightb | ;add var weightb to register bx |
| idiv bx | ;multiply ax by bx |
| | |
| mov a, 0 | ;move the integer to var a |
| mov weighta, 1 | ;move the integer to var weighta |
| mov ax, weighta | ;move the variable weighta into ax |
| imul a | ;signed multiply ax by var a |
| mov SWORD PTR result, ax | ;place result of ax into var result |
| mov b, 0 | ;move the integer into var b |
| mov weightb, 1 | ;move the integer into var weightb |
| mov ax, weightb | |
| imul b | ;signed multiply ax by var b |
| add ax, SWORD PTR result | ;add the var result to register ax |
| mov bx, weighta | ;move the var weight a into register bx |
| add bx, weightb | ;add var weightb to register bx |
| idiv bx | ;multiply ax by bx |
| | |
| mov a, 0 | ;move the integer to var a |
| mov weighta, 1 | ;move the integer to var weighta |
| mov ax, weighta | ;move the variable weighta into ax |
| imul a | ;signed multiply ax by var a |
| mov SWORD PTR result, ax | ;place result of ax into var result |
| mov b, 0 | ;move the integer into var b |
| mov weightb, 1 | ;move the integer into var weightb |
| mov ax, weightb | |
| imul b | ;signed multiply ax by var b |
| add ax, SWORD PTR result | ;add the var result to register ax |
| mov bx, weighta | ;move the var weight a into register bx |
| add bx, weightb | ;add var weightb to register bx |
| idiv bx | ;multiply ax by bx |
| | |
| mov a, 0 | ;move the integer to var a |
| mov weighta, 1 | ;move the integer to var weighta |
| mov ax, weighta | ;move the variable weighta into ax |
| imul a | ;signed multiply ax by var a |
| mov SWORD PTR result, ax | ;place result of ax into var result |
| mov b, 0 | ;move the integer into var b |
| mov weightb, 1 | ;move the integer into var weightb |
| mov ax, weightb | |
| imul b | ;signed multiply ax by var b |
| add ax, SWORD PTR result | ;add the var result to register ax |
| mov bx, weighta | ;move the var weight a into register bx |
| add bx, weightb | ;add var weightb to register bx |
| idiv bx | ;multiply ax by bx |
| | |
| cpuid | ;force serialization of instructions |
| rdtsc | ;get the cycles |
| sub eax, subtime | ;subtract the stored from the current result |
| invoke displaymsg, addr mess7, mess7len | ;display message 7 |

```
        call PutDec
        .exit
main endp
```

```
;display the quotient of the division to stdout
```

```
;*****
;Function: displaymsg
;Description:
;   This function takes a pointer to a string
;   as input and the length of the string and
;   prints the result to stdout. It does this
;   by using the OS int 21h and function 040h.
;   BX holds the device (1), cl holds the
;   length of string, dx holds the address of
;   the string to be displayed. Flags and
;   the 32-bit registers are pushed and popped
;   of the stack.
;Input:
;   m - a pointer to the string which is to be
;       displayed to stdout
;   l - an short integer indicating the length
;       of the string to display
;Output:
;   None
;Return:
;   None
;*****
displaymsg proc near c uses ax bx cx dx, m:word,l:byte
    pushf
    pushad
    mov ah, 040h
    mov bx, 1
    mov cl, l
    shl ch, 8
    mov dx, m
    int 21h
    popad
    popf
    ret
displaymsg endp
```

```
end
```