Project 3

Hazael Magino

3/25/24

The Goal of this program is to enhance the development of file I/o and arithmetic calculations along with enhance one's ability with string manipulation in low level architecture (x86).

Milestone 1

In this program the desired goal was to show the implementation of the following tasks

- Be able to open, read, and write from a file.
- Be able to separate different data types from a file.
- Output specific data types to each specific type file.
- *Largest floating-point number
- Summation of integers
- Print the number of data types read from a file.

File I/0

- For this part of the programmed I enhanced my knowledge from project 2 by utilizing system calls to allow me to be able to open a file and read the necessary contents. In addition to that I was also required to create my own files and input data into those files through string manipulation.

```
_start:

    jmp open
open:
    ;open text file

    mov ebx, filename           ; Getting memory address of the txt file
    mov eax, 5                  ; syscall of 5 dictates to opening of a
file
    mov ecx, 0                  ; set ecx to read only mode

    int 0x80

    jmp read

read:
    ;Here we first read all the info to a buffer

    mov ebx, [readp]            ;allows to stores the file descriptor into
the memory location
    mov eax, 3                  ;sys_read call
```

Project 3

Hazael Magino

3/25/24

```
    mov ebx, eax                ; set readp as input for read operation
    mov ecx, genbuf             ;load memory address of where to store data
    mov edx, ltoread              ;Amount of bytes to read

    int 0x80


    mov eax, 0                  ; this will be the length of in the buffer
    xor ebx, ebx                ; initialize count digits
    xor ecx, ecx                ;initalize to all zeros
    cld                         ;set loadsb to interpret data from left to
 right of the genbuf
    lea esi, genbuf              ;Set si to the genbuf first value will be
 the the number of entries in file
    jmp getfileLen

```

Source used: [assembly - Reading text file line by line NASM - Stack Overflow]

*One debugging issue that was not managed to be solved was why genbuf could not be populated with the file contents. Similar procedure was used on the previous project but expected results were not the same

Print Number of Data types

- For printing printing Number of data types string manipulation was needed because we needed to find a way to process each type. In addition proper register management would be needed in order to execute with enough resources. One way I managed to do this was inc whenever I came across a space or newline. This value would then be kept in a buffer which would make the process of not using any C functions to print out the value when we can utilize system calls. Below is a representation of this with int data type:

```
outputentryint:

    ; Store the total in EDX into calcinstumbuf
    mov [calcinstumbuf], edx

    ; Store the value of entries in ah into intnumentrybuf
    mov [intnumentrybuf], ah

    ; Write the content of intsumtxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, intsumtxt
```

Project 3

Hazael Magino

3/25/24

```
    mov edx, intlennum
    int 0x80

    ; Write the content of intnumentrybuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, intnumentrybuf
    mov edx, 128
    int 0x80

    ; Write the content of inttlttxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, inttlttxt
    mov edx, 128
    int 0x80

    ; Write the content of calcinstumbuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, calcinstumbuf
    mov edx, 128                ; Number of bytes to write (32 because the max
number of bytes in 1000 is 4)
    int 0x80

    ; Clear registers
    xor ebx, ebx
    xor ecx, ecx
    xor edx, edx
    xor eax, eax

    ; Load the address of floatbuf into SI
    mov si, floatbuf

    ; Jump to the label 'calcentryflt'
    jmp calcentryflt
```

Separate Data Types

- This was the main point of the program and most of it came down to string manipulation and comparison. Utilizing the ascii table extensively, made this portion doable as implementing

Project 3

Hazael Magino

3/25/24

simple logic would not suffice. My thought process for this portion came down to the complexity with each type. The strings were the least complicated so there would be less logic for it. Integers were a little complex when involving negative numbers but that would be simple in terms of dealing with. Floats were the most complex so writing the most “if” statements would be the most logical and intuitive because they can be both integers and floats. I first detirmed if the value of the data type had a negative. If so I went to decide if it were an float or an integer. The easiest way to tell if a data point was negative was if the 2nd or 3rd byte was a decimal point or a period, if so act accordingly. Below is how I initially determined the data types.

```
eperate:

    ;Here we plan to copy data from the general buffer to the specific buffers based on the data itself
    ;string buffer, intger buffer, and floating point buffer

    lodsb                       ; load character into Al register from genbuf

    cmp al, 10                  ;Check if it is a new line

    je seperate

    test al, al                 ;Check if we've reached the end of the file (null terminator)

    jz outputfile               ;reached end of file print output files of whatever values have been found

    cmp al, 32                  ;Check if it is a space line

    je seperate

    cmp al, 45                  ; cmp the ascii values and use " - " that as
    ;value will either be a float or an integer if true

    je ifnegative               ; if there is a negative value then we know it can be an integer or float

    mov [tempbuf], al           ;hold to detirmine if the next value is a float

```

Project 3

Hazael Magino

3/25/24

```
    lodsb

    cmp al, 46                  ;check if the next value is a "."
indicating a float value

    je storeflt

    jmp storestr


ifnegative:
    ;detirmine if the string is a negative int or negative float

    push ax

    xor al, al                  ;prepare for the next value

    lodsb                       ;load next character into al register

    mov [negvalbuf1], al        ;store 2ndvalue on buffer

    lodsb

    mov [negvalbuf2], al        ;store 3rd value on buffer

    cmp al, 46

    je negfloat                 ;check if the next value is a "."
indicating a float value



    jmp negint                  ;if it is not a "." then we know that the
next character will be a number
negint:
```

Largest Floating point

- In my code I was not able to implement this as I could not figure out how to really sort the values in a conventional way like bubble sort. This would probably the most likely way to do out of all the methods as it is the least complicated in such a low level format. With this

implementation you would have the ability to sort the values in such a way where the first value in the buffer would be the smallest value and the last value would be the greatest.

Integer Summation

- With this implementation, I developed a similar way to accumulate the sum similar to counting the entries in a file. However this time I had to account for the fact that I also needed to keep track of the sum across both addition and subtraction. Therefore with subtraction steps and labels had to be replicated in order to process the values in the same fashion but with the respective operation. Here is the subtraction portion of the code:

```
subint:
    ;increment entry value and decrement total by integer CHANGEEEE
    lodsb

    cmp al, 0                   ;See if we have reached the null
terminator
    je calcsubsumpower
    cmp al, 10                  ;Check if we have a new line
    je calcsubsumpower
    cmp al, 32                  ;Check if we have a space
    je countsubspace


    push ax                     ;accumulate the value to the previous sum
    inc bx                      ;this helps with detirmining the powers
    jmp precalcintsum

countsubspace:

    inc ah
    jmp calcsubsumpower

calcsubsumpower:
    ;Here this detimines the powers of 10 needed to get the correct sum

    cmp bx, 4
    je subcalc4
    cmp bx, 3
    je subcalc3
    cmp bx, 2
    je subcalc2
    cmp bx, 1
    je subcalc1
```

Project 3

Hazael Magino

3/25/24

```
subcalc4:
    ;this is for if we have 4 digits in our number
    ;last number on the stack is the highest value

    push ax
    sub edx, eax
    push ax
    mov cx, 10
    mul cx
    sub edx, ecx
    push ax
    mov cx, 100
    mul cx
    sub edx, ecx
    push ax
    mov cx, 1000
    mul cx
    sub edx, ecx

    jmp precalcintsum

subcalc3:
    ;this is for if we have 3 digits in our number

    push ax
    sub edx, eax
    push ax
    mov cx, 10
    mul cx
    sub edx, ecx
    push ax
    mov cx, 100
    mul cx
    sub edx, ecx

    jmp precalcintsum

subcalc2:
    ;this is for if we have 2 digits in our number

    push ax
```

Project 3

Hazael Magino

3/25/24

```
        sub dx, ax
        push ax
        mov cx, 10
        mul cx
        sub edx, ecx

        jmp precalcintsum

    subcalc1:
        ;this is for if we have 2 digits in our number

        push ax
        sub dx, ax

        jmp precalcintsum

```

Helpful sources:

These are some of the external resources I utilized to formulate my code:

x86 - Help with assembly code (convert string to int) - Stack Overflow

assembly - Reading text file line by line NASM - Stack Overflow

https://youtu.be/oRjLQ8_aPZ8?si=QljSoGagBY14wfoM

assembly-nasm-codes/library/syscall_table.md at main · GabriOliv/assembly-nasm-codes · GitHub

x86 16 - Assembly 8086 - copy one buffer to another - Stack Overflow

As well as some insight from chat gpt in correct operand usage cases

Code

Here is the full code implementation:

```
section .data:

    filename db "./afs/umbc.edu/users/h/a/hazaelm1/home/CMPE/310/proj3/testfile.txt" ;test file being used
    outputint dd 'project3_int.out'
    outputflt dd 'project3_float.out'
    outputstr dd 'project3_string.out'
```

Project 3

Hazael Magino

3/25/24

```
    filelentxt dd 'The length of the file is: ', 10
    filelennum equ $ - filelentxt

    intsumtxt db 'The number of integers in the file is: ', 10
    intlennum equ $ - intsumtxt
    inttlttxt dd 'The total sum of integers is in the file: ', 10
    inttltnum equ $ - inttlttxt

    strsumtxt dd 'The number of strings in the file is: ', 10
    fltsumtxt dd 'The number of floats in the file is: ', 10
    finish dd 'Output files have been created and written', 10



section .bss

    genbuf resb 1024              ;Here we store the file values
    readp resb 4

    filelen resb 32               ; this will be the length of file we store
    stringbuf resb 1024           ;max amount of strings to store on buffer
    floatbuf resb 1024            ; max amount of floating point strings numbers
to take in
    intbuf resb 1024              ; max amount of integers to read from fil

    intnumentrybuf resb 8         ;The amount of entries within the int buf
    stringnumentrybuf resb 8      ;The amount of entries within the string buf
    tempbuf resb 1                ; this holds the temporary value being loaded
into the al register
    negvalbuf1 resb 2
    negvalbuf2 resb 2
    saveintlo resb 128            ;This is to save the location of the address in
the intbuffer
    ;before we move the register to access another buffer if necessary
    savestrlo resb 128            ;This is to save the location of the address in
the strbuffer
    ;before we move the register to access another buffer if necessary
    saveflolo resb 128            ;This is to save the location of the address in
the floatbuffer
    ;before we move the register to access another buffer if necessary
```

Project 3

Hazael Magino

3/25/24

```
    calcinstumbuf resb 8          ; This will hold the total sum value of the
inbuf to output
    fltnumentrybuf resb 8         ;The amount of entries within the flt buf
    ltoread resb 1024 ;for the file name
section .text

    global _start


_start:

    jmp open
open:
    ;open text file

    mov ebx, filename             ; Getting memory address of the txt file
    mov eax, 5                    ; syscall of 5 dictates to opening of a file
    mov ecx, 0                    ; set ecx to read only mode

    int 0x80

    jmp read

read:
    ;Here we first read all the info to a buffer

    mov ebx, [readp]              ;allows to stores the file descriptor into the
memory location
    mov eax, 3                    ;sys_read call
    mov ebx, eax                 ; set readp as input for read operation
    mov ecx, genbuf               ;load memory address of where to store data
    mov edx, ltoread                ;Amount of bytes to read

    int 0x80


    mov eax, 0                    ; this will be the length of in the buffer
    xor ebx, ebx                  ; initialize count digits
    xor ecx, ecx                  ;initalize to all zeros
    cld                           ;set loadsb to interpret data from left to right
of the genbuf
```

Project 3

Hazael Magino

3/25/24

```
    lea esi, genbuf                ;Set si to the genbuf first value will be the
the number of entries in file
    jmp getfileLen


getfileLen:

    lodsb                          ;utilize sb because the number can 32 bits long
or up to 4bytes (1000)
    cmp al, 0                      ;check if we have reached a null terminator in
the buffer
    je calcPowers                  ;We have the value full value of ax and have
encountered newline
    cmp al, 10                     ;check if it is a newline
    je calcPowers
    cmp al, 32                     ;Check if it is a space
    je calcPowers                  ;We have the value full value of ax and have
encountered a null or new line

    add bx, 1                      ;increment for powers
    jmp transcribe

transcribe:
    ;for converting the values from ascii to integers


    sub al, 48                     ; conversion to integer value to acculmulate the
total sum to be used
    push ax                        ;push to stack to keep track

    jmp getfileLen
calcPowers:
    ;here we detirmine the power of each digit on the stack based on the amount
    ;of items that were pushed (cx)

    xor edx, edx

    cmp bx, 4
    je fourpowers
    cmp bx, 3
    je threepowers
    cmp bx, 2
```

Project 3

Hazael Magino

3/25/24

```
    je twopowers
    cmp bx, 1
    je onepower

fourpowers:
    ;this is for if we have 4 digits in our number
    ;last number on the stack is the highest value
    mov edx, 0
    mov ecx, 0
    mov eax, 0


    push ax
    add dx, ax
    push ax
    mov cx, 10
    mul cx
    add edx, ecx
    push ax
    mov cx, 100
    mul cx
    add edx, ecx
    push ax
    mov cx, 1000
    mul cx
    add edx, ecx

    mov [filelen], edx
    ;now we output this
    jmp printfilesum

threepowers:
    ;this is for if we have 3 digits in our number
    ;last number on the stack is the lesser value
    mov edx, 0
    mov ecx, 0
    mov eax, 0
    push ax
    add dx, ax
    push ax
    mov cx, 10
    mul cx
```

Project 3

Hazael Magino

3/25/24

```
    add edx, ecx
    push ax
    mov cx, 100
    mul cx
    add edx, ecx

    mov [filelen], edx
    ;now we output this
    jmp printfilesum


twopowers:
    ;this is for if we have 2 digits in our number
    ;last number on the stack is the lesser value
    mov edx, 0
    mov ecx, 0
    mov eax, 0
    push ax
    add dx, ax
    push ax
    mov cx, 10
    mul cx
    add edx, ecx


    mov [filelen], edx
    ;now we output this
    jmp printfilesum


onepower:
    ;this is for if we have 1 digits in our number
    ;last number on the stack is the lesser value
    mov edx, 0
    mov ecx, 0
    mov eax, 0
    push ax
    add edx, edx

    mov [filelen], edx
    ;now we output this
    jmp printfilesum
```

Project 3

Hazael Magino

3/25/24

```
printfilesum:
    ;here we output the total number of entries in the file

    mov eax, 4
    mov ebx, 1
    mov ecx, filelentxt          ; message for amount of entries
    mov edx, filelennum          ; move the total value of what in the sum to
filelentxt is
    int 0x80

    mov eax, 4                   ;call for sys_write
    mov ebx, 1                   ;set readp for standard output
    mov ecx, filelen             ; load data read to be written to output
    mov edx, 32                  ; Number of bytes to write (32 because the max
number of bytes in 1000 is 4)

    int 0x80
    xor eax, eax
    mov ebx, ebx
    mov ecx, ecx
    mov edx, edx

    cld                          ;read from left to right
    lea esi, genbuf

    jmp seperate
seperate:

    ;Here we plan to copy data from the general buffer to the specific buffers
based on the data itself
    ;string buffer, intger buffer, and floating point buffer

    lodsb                        ; load character into Al register from genbuf

    cmp al, 10                   ;Check if it is a new line

    je seperate

    test al, al                  ;Check if we've reached the end of the file
(null terminator)
```

Project 3

Hazael Magino

3/25/24

```
    jz outputfile                ;reached end of file print output files of
whatever values have been found

    cmp al, 32                   ;Check if it is a space line

    je seperate

    cmp al, 45                   ; cmp the ascii values and use " - " that as
    ;value will either be a float or an integer if true

    je ifnegative                ; if there is a negative value then we know it
can be an integer or float

    mov [tempbuf], al            ;hold to detirmine if the next value is a float

    lodsb

    cmp al, 46                   ;check if the next value is a "." indicating a
float value

    je storeflt

    jmp storestr


ifnegative:
    ;detirmine if the string is a negative int or negative float

    push ax

    xor al, al                   ;prepare for the next value

    lodsb                        ;load next character into al register

    mov [negvalbuf1], al         ;store 2ndvalue on buffer

    lodsb

    mov [negvalbuf2], al         ;store 3rd value on buffer

    cmp al, 46
```

Project 3

Hazael Magino

3/25/24

```
    je negfloat                 ;check if the next value is a "." indicating a
float value



    jmp negint                  ;if it is not a "." then we know that the next
character will be a number
negint:
    ;this is a special case where we identified that the value is a negative int
    ;here we must remember to access the previous value we found before we
detimrned
    ;if the next byte was a "." or an intger value

    mov bx, [negvalbuf2]           ; here we move the previous value(3)
identified into the high register of ebx (bh)

    mov ah, [negvalbuf1]           ; here we move the previous value(2)
identified into the high register of eax (ah)

    push ax

    mov di, [intbuf]

    sub al, 48

    mov [di], al

    inc di                      ;manually increment di so that when we input the
next value we are not overwitting the previous value

    sub ah, 48

    mov [di], ah

    inc di                      ;manually increment di so that when we input the
next value we are not overwitting the previous value

    sub ah, 48

    mov [di], bh
```

Project 3

Hazael Magino

3/25/24

```
    inc di                      ;manually increment di so that when we input the
next value we are not overwitting the previous value

    mov [intbuf], di

    jmp seperate

negfloat:
    ;this is a special case where we identified that the value is a negative
float
    ;value and must be dealt with in the specific case

    mov bh, [negvalbuf2]          ;this will be the 3rd value

    mov ah, [negvalbuf1]          ; this will be the 2nd value

    push ax                     ; this will be the first value we pushed to the
stack

    mov di, [floatbuf]           ; get the address of the floatbuffer

    sub al, 48                  ;convert to int

    mov [di], al                ;put it in the buffer

    inc di                      ;manually inc buffer

    sub al, 48

    mov [di], ah

    inc di

    sub bh, 48

    mov [di], bh

    mov [floatbuf], di

    lodsb                       ;load next float
```

Project 3

Hazael Magino

3/25/24

```
    cmp al, 10                  ; if a new line is detected then we have reached
the end

    je seperate

    cmp al, 32                  ; if a space is detected then we have reached
the end

    je seperate

    test al, al

    jz outputfile               ; (finish when final output has been made)

    jmp storeflt
storestr:
    ; Store the string being pointed at to the strbuffer

    mov ah, [tempbuf]             ; Move the value in tempbuf to AH register
    mov di, [stringbuf]           ; Load the address of savestrlo into DI
register
    xor ebx, ebx                ; Clear EBX register (set it to zero)
    mov bh, 65                  ; Set BH register to 65 (ASCII value for 'A')

    ; Compare AL (the lower 8 bits of AX) with BH
    cmp al, bh
    jl storeInt                 ; If AL < BH, jump to storeInt

    ; Otherwise, store the characters in the buffer
    mov [di], ah                ; Store the value in AH at the address in DI
    inc di                      ; Increment DI (point to the next memory
location)
    mov [di], al                ; Store the value in AL at the new address in DI
    inc di                      ; Increment DI again
    mov [stringbuf], di         ; Update the value of savestrlo with the new
address
    jmp seperate                ; Jump to the label 'seperate'


storeInt:
    ; Store the int being pointed at to the intbuffer
```

Project 3

Hazael Magino

3/25/24

```
    ; Move the value in tempbuf to AH register
    mov ah, [tempbuf]

    ; Load the address of saveintlo into DI register
    mov di, [intbuf]

    ; Clear BL register (set it to zero)
    xor bl, bl

    ; Set BL register to 57 (ASCII value for '9')
    mov bl, 57

    ; Compare AL (the lower 8 bits of AX) with BL
    cmp al, bl
    jg storestr                 ; If AL > BL, jump to storestr

    ; Otherwise, perform integer conversion
    sub ah, 48                  ; Convert ASCII digit to actual value
    mov [di], ah                ; Store the value in AH at the address in DI
    inc di                      ; Increment DI (point to the next memory
location)
    sub al, 48                  ; Convert ASCII digit to actual value
    mov [di], al                ; Store the value in AL at the new address in DI
    inc di                      ; Increment DI again
    mov [intbuf], di            ; Update the value of saveintlo with the new
address
    jmp seperate                ; Jump to the label separate



storeflt:
    ;store the floating point number in this step


    ; Move the value in tempbuf to AH register
    mov ah, [tempbuf]

    ; Load the address of saveflolo into DI register
    mov di, [floatbuf]

    ; Clear BL register (set it to zero)
    xor bl, bl
```

Project 3

Hazael Magino

3/25/24

```
    ; Convert ASCII digit to actual value by subtracting 48
    sub ah, 48
    mov [di], ah                ; Store the value in AH at the address in DI
    inc di                      ; Increment DI (point to the next memory
location)
    sub al, 48
    mov [edi], al                ; Store the value in AL at the new address in
DI
    inc di                      ; Increment DI again
    mov [floatbuf], di          ; Update the value of saveflolo with the new
address

    lodsb                       ; Load the string pointed to by SI

    ; Check if a new line (ASCII 10) is detected
    cmp al, 10
    je seperate                 ; If true, jump to the label 'separate'

    ; Check if a space (ASCII 32) is detected
    cmp al, 32
    je seperate                 ; If true, jump to the label 'separate'

    ; Test if AL is zero (end of string)
    test al, al
    jz outputfile               ; If AL is zero, jump to the label 'outputfile'

    jmp storeflt                ; Otherwise, jump to the label 'storeflt'

outputfile:
    ;here we create corresponding files for specific buffers

    ;output ints
    mov eax, 8
    mov ebx, outputint          ;int text file being created
    mov ecx, 0777               ;0777 is permissions of the user to read write
and execute
    int 0x80

    mov [readp], eax

    mov edx, 1024
```

Project 3

Hazael Magino

3/25/24

```
    mov ecx, intbuf             ;intbuffer with values to write
    mov ebx, [readp]            ;file descriptor
    mov edx, 4                  ;sys call for write
    int 0x80

    mov eax, 6                  ;close the file
    mov ebx, [readp]
    int 0x80

    ;output strings
    mov eax, 8
    mov ebx, outputstr          ;int text file being created
    mov ecx, 0777               ;0777 is permissions of the user to read write
and execute
    int 0x80

    mov [readp], eax

    mov edx, 1024
    mov ecx, stringbuf          ;intbuffer with values to write
    mov ebx, [readp]            ;file descriptor
    mov edx, 4                  ;sys call for write
    int 0x80

    mov eax, 6                  ;close the file
    mov ebx, [readp]
    int 0x80

    ;output floats
    mov eax, 8
    mov ebx, outputflt          ;int text file being created
    mov ecx, 0777               ;0777 is permissions of the user to read write
and execute
    int 0x80

    mov [readp], eax

    mov edx, 1024
    mov ecx, floatbuf           ;intbuffer with values to write
    mov ebx, [readp]            ;file descriptor
    mov edx, 4                  ;sys call for write
    int 0x80
```

Project 3

Hazael Magino

3/25/24

```
    mov eax, 6                  ;close the file
    mov ebx, [readp]

    int 0x80

    ;print a message indicating files have been created

    mov eax, 4
    mov ebx, 1
    mov ecx, finish
    mov edx, 64
    int 0x80

    ;Calculate final int sum and number of entries
    lea si, intbuf            ;move si to integer buffer
    xor edx, edx
    xor ebx, ebx
    mov ecx, 0
    jmp precalcintsum

calcentryflt:
    ;Calculate the entries of float values

    lodsb                       ; Load the string pointed to by DS

    cmp al, 32
    je incentryflt              ; If AL is equal to 32, jump to the label
'incentryflt'

    cmp al, 10                  ; Compare AL with 10 (ASCII for newline)
    je incentryflt              ; If AL is equal to 10, jump to the label
'incentryflt'

    cmp al, 0                   ; Compare AL with 0 (end of string)
    je outputentryflt           ; If AL is zero, jump to the label
'outputentryflt'

    ; Otherwise, jump to the label 'calcentryflt'
    jmp calcentryflt
```

Project 3

Hazael Magino

3/25/24

```
outputentryflt:
    ;output the number of floats

    ; Store the value in CX into fltnumentrybuf
    mov [fltnumentrybuf], cx

    ; Write the content of fltsumtxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, fltsumtxt
    mov edx, 32
    int 0x80

    ; Write the content of fltnumentrybuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, fltnumentrybuf
    mov edx, 128                ; Number of bytes to write (32 because the max
number of bytes in 1000 is 4)
    int 0x80

    ; Clear registers
    xor ebx, ebx
    xor ecx, ecx
    xor edx, edx
    xor eax, eax

    ; Load the address of stringbuf into SI
    lea esi, stringbuf

    ; Jump to the label 'calcentrystr'
    jmp calcentrystr

    ; (Note: The label 'calcentrystr' is referenced but not defined in this
snippet)


incentryflt:

    inc cx                      ;increment cx to keep track of how many spaces
or newlines (values) we have
    jmp calcentryflt
```

Project 3

Hazael Magino

3/25/24

```
calcentrystr:
    ;Calculate the number of string entries

    lodsb                       ; Load the string pointed to by DS:SI into the
buffer

    cmp al, 10                  ; Compare AL (the lower 8 bits of AX) with 10
(ASCII for newline)
    je incentrstr               ; If AL is equal to 10, jump to the label
'incentrstr'

    cmp al, 32                  ; Compare AL with 32 (ASCII for space)
    je incentrstr               ; If AL is equal to 32, jump to the label
'incentrstr'

    cmp al, 0                   ; Compare AL with 0 (end of string)
    je outputentrystr           ; If AL is zero, jump to the label
'outputentrystr'

    ; Otherwise, jump to the label 'calcentrystr'
    jmp calcentrystr

incentrstr:

    inc dx                      ;increment dx to help us keep track of how many
entries there are
    jmp calcentrystr

outputentrystr:
    ;output the number of string entries

    ; Store the value in DX into stringnumentrybuf
    mov [stringnumentrybuf], dx

    ; Write the content of strsumtxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, strsumtxt
    mov edx, 64
    int 0x80
```

Project 3

Hazael Magino

3/25/24

```
    ; Write the content of stringnumentrybuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, stringnumentrybuf
    mov edx, 128                ; Number of bytes to write (32 because the max
number of bytes in 1000 is 4)
    int 0x80

    ; Exit the program
    mov eax, 1                  ; System call number for sys_exit
    int 0x80                    ; Call the kernel


outputentryint:

    ; Store the total in EDX into calcinstumbuf
    mov [calcinstumbuf], edx

    ; Store the value of entries in ah into intnumentrybuf
    mov [intnumentrybuf], ah

    ; Write the content of intsumtxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, intsumtxt
    mov edx, intlennum
    int 0x80

    ; Write the content of intnumentrybuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, intnumentrybuf
    mov edx, 128
    int 0x80

    ; Write the content of inttlttxt to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, inttlttxt
    mov edx, 128
    int 0x80
```

Project 3

Hazael Magino

3/25/24

```
    ; Write the content of calcinstumbuf to standard output
    mov eax, 4
    mov ebx, 1
    mov ecx, calcinstumbuf
    mov edx, 128                ; Number of bytes to write (32 because the max
number of bytes in 1000 is 4)
    int 0x80

    ; Clear registers
    xor ebx, ebx
    xor ecx, ecx
    xor edx, edx
    xor eax, eax

    ; Load the address of floatbuf into SI
    lea esi, floatbuf

    ; Jump to the label 'calcentryflt'
    jmp calcentryflt

precalcintsum:

    lodsb                       ;load the first int in the int buffer

    cmp al, 45                  ;See if the value is negative first
    je subint
    cmp al, 0                   ;See if we have reached the null terminator
    je outputentryint
    cmp al, 10                  ;Check if we have a new line
    je calcsumpower
    cmp al, 32                  ;Check if we have a space
    je countspace

    sub al, 48                  ;calculate the value of the string int
    push ax                     ;accumulate the value to the previous sum
    inc bx                      ;this helps to detirmine the power figure
    jmp precalcintsum

calcsumpower:
    ;Here this detimines the powers of 10 needed to get the correct sum
    cmp bx, 4
```

Project 3

Hazael Magino

3/25/24

```
    je calc4
    cmp bx, 3
    je calc3
    cmp bx, 2
    je calc2
    cmp bx, 1
    je calc1

calc1:
    ;this is for if we have 2 digits in our number

    push ax
    add edx, eax

    jmp precalcintsum


calc2:
    ;this is for if we have 2 digits in our number

    push ax
    add edx, eax
    push ax
    mov cx, 10
    mul cx
    add edx, ecx


    jmp precalcintsum


calc3:
    ;this is for if we have 3 digits in our number

    push ax
    add edx, eax
    push ax
    mov cx, 10
    mul cx
    add edx, ecx
    push ax
    mov cx, 100
```

Project 3

Hazael Magino

3/25/24

```
    mul cx
    add edx, ecx

    jmp precalcintsum


calc4:
    ;this is for if we have 4 digits in our number
    ;last number on the stack is the highest value

    push ax
    add edx, eax
    push ax
    mov cx, 10
    mul cx
    add edx, ecx
    push ax
    mov cx, 100
    mul cx
    add edx, ecx
    push ax
    mov cx, 1000
    mul cx
    add edx, ecx

    jmp precalcintsum

countspace:

    ;help us identify how many numbers we have
    inc ah
    jmp precalcintsum

subint:
    ;increment entry value and decrement total by integer CHANGEEEE
    lodsb

    cmp al, 0                   ;See if we have reached the null terminator
    je calcsubsumpower
    cmp al, 10                  ;Check if we have a new line
    je calcsubsumpower
    cmp al, 32                  ;Check if we have a space
```

Project 3

Hazael Magino

3/25/24

```
    je countsubspace


    push ax                     ;accumulate the value to the previous sum
    inc bx                      ;this helps with detirmining the powers
    jmp precalcintsum

countsubspace:

    inc ah
    jmp calcsubsumpower

calcsubsumpower:
    ;Here this detimines the powers of 10 needed to get the correct sum

    cmp bx, 4
    je subcalc4
    cmp bx, 3
    je subcalc3
    cmp bx, 2
    je subcalc2
    cmp bx, 1
    je subcalc1

subcalc4:
    ;this is for if we have 4 digits in our number
    ;last number on the stack is the highest value

    push ax
    sub edx, eax
    push ax
    mov cx, 10
    mul cx
    sub edx, ecx
    push ax
    mov cx, 100
    mul cx
    sub edx, ecx
    push ax
    mov cx, 1000
    mul cx
    sub edx, ecx
```

Project 3

Hazael Magino

3/25/24

```
    jmp precalcintsum

subcalc3:
    ;this is for if we have 3 digits in our number

    push ax
    sub edx, eax
    push ax
    mov cx, 10
    mul cx
    sub edx, ecx
    push ax
    mov cx, 100
    mul cx
    sub edx, ecx

    jmp precalcintsum

subcalc2:
    ;this is for if we have 2 digits in our number

    push ax
    sub dx, ax
    push ax
    mov cx, 10
    mul cx
    sub edx, ecx

    jmp precalcintsum

subcalc1:
    ;this is for if we have 2 digits in our number

    push ax
    sub dx, ax

    jmp precalcintsum
```

Project 3

Hazael Magino

3/25/24

Output:

*As specified in the beginning file input could not be read despite similar code formats being used in the last project for this project. Output below is the only output of the code: