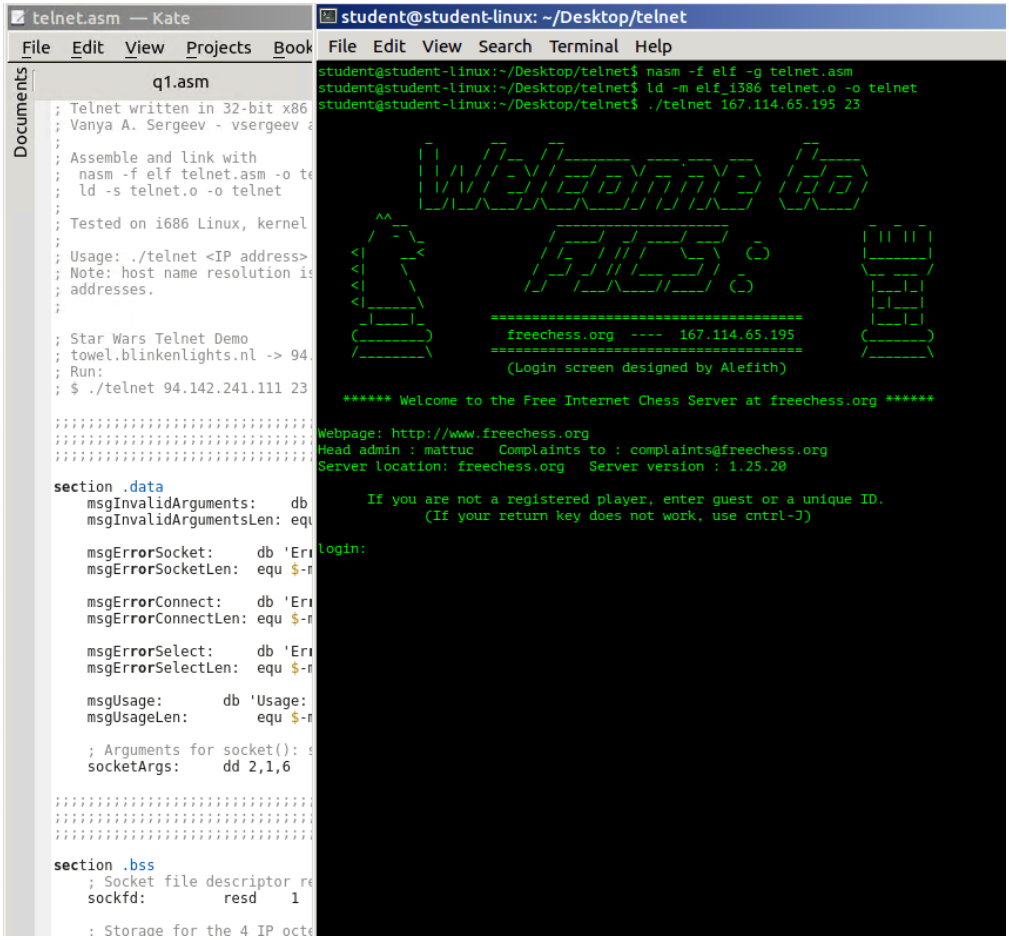
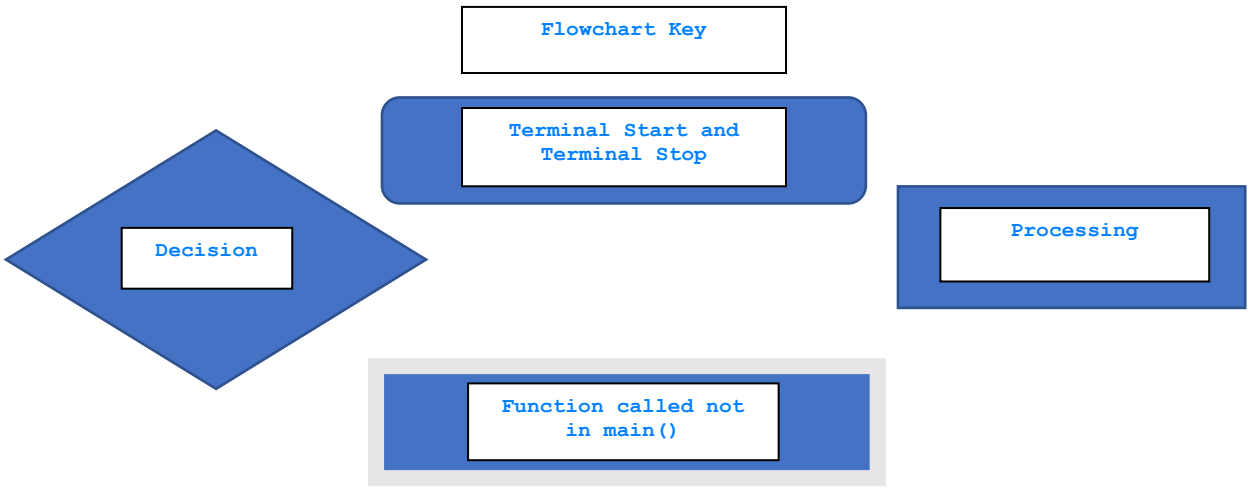
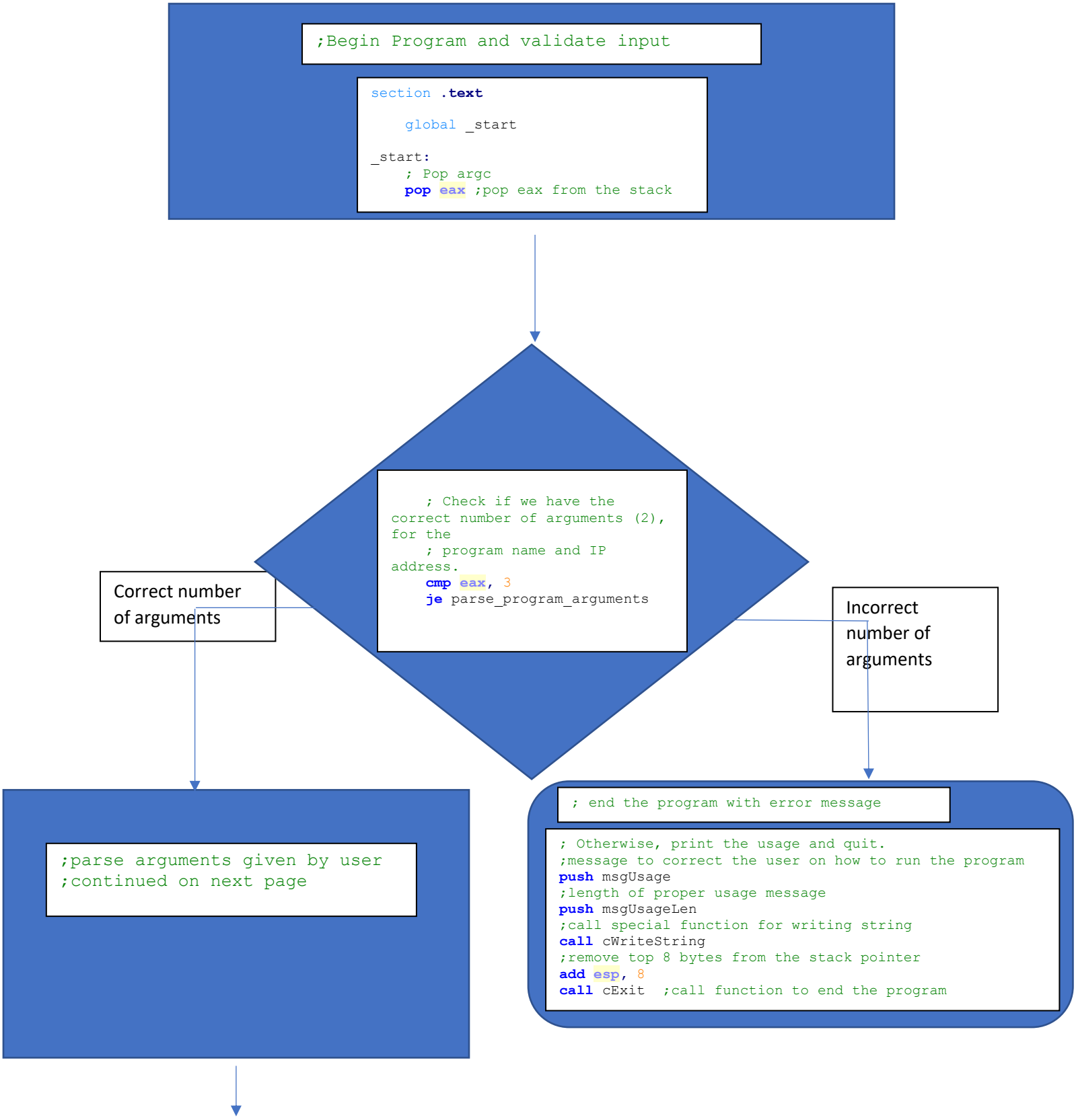
Kyle AuBuchon CYBV471 Telnet Final Project





Initialize Data and allocate memory for variables

```
section .data
    msgInvalidArguments:    db 'Invalid IP address or port supplied!',10,0
    msgInvalidArgumentsLen: equ $-msgInvalidArguments

    msgErrorSocket:     db 'Error creating socket!',10,0
    msgErrorSocketLen:  equ $-msgErrorSocket

    msgErrorConnect:    db 'Error connecting to server!',10,0
    msgErrorConnectLen: equ $-msgErrorConnect

    msgErrorSelect:     db 'Error with select()!',10,0
    msgErrorSelectLen:  equ $-msgErrorSelect

    msgUsage:       db 'Usage: ./telnet <IP address> <port>',10,0
    msgUsageLen:        equ $-msgUsage

    ; Arguments for socket(): socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    socketArgs:     dd 2,1,6

section .bss
    ; Socket file descriptor returned by socket()
    sockfd:         resd    1

    ; Storage for the 4 IP octets
    ipOctets        resb    4

    ; Storage for the connection port represented in one 16-bit word
    ipPort          resw    1

    ; Arguments for connect():
    ;   connect(sockfd, serverSockaddr, serversockaddrLen);
    connectArgs     resd    3

    ; The read file descriptor array for select()
    masterReadFdArray   resb    128
    checkReadFdArray    resb    128
    readFdArrayLen      equ 128

    ; sockaddr_in structure that needs to be filled in for the
    ; connect() system call.
    ;   struct sockaddr_in {
    ;       short            sin_family;
        ;       unsigned short   sin_port;
        ;       struct in_addr   sin_addr;
        ;       char             sin_zero[8];
    ;   };
    serverSockaddr      resb    (2+2+4+8)
    serverSockaddrLen   equ 16

    ; Read buffer for reading from stdin and the socket
    readBuffer      resb    1024
    readBufferLen       resd    1
    readBufferMaxLen    equ 1024
```


;Begin Program and validate input
section .text
global _start
_start:
; Pop argc
pop eax ;pop eax from the stack
; Check if we have the correct number of arguments (2), for the
; program name and IP address.
cmp eax, 3
je parse_program_arguments
Correct number of arguments
Incorrect number of arguments
;parse arguments given by user
;continued on next page
; end the program with error message
; Otherwise, print the usage and quit.
;message to correct the user on how to run the program
push msgUsage
;length of proper usage message
push msgUsageLen
;call special function for writing string
call cWriteString
;remove top 8 bytes from the stack pointer
add esp, 8
call cExit ;call function to end the program

;Getting user input and converting string to decimals

```
; Set the direction flag to increment, so edi/esi are INCREMENTED

; with their respective load/store instructions.
cld ;clear direction flag

; Pop the program name string
pop eax ;pop eax from the stack

;;; Convert the port and IP address strings to numbers ;;;

; Next on the stack is the IP address
; Convert the IP address string to four byte sized octets.
call cStrIP_to_Octets   ;call function to convert IP into individual octets
add esp, 4  ;remove top 4 bytes from the stack pointer
```

;cStrip_to_Octets function is called

```
;
; cStrIP_to_Octets
;   Parses an ASCII IP address string, e.g. "127.0.0.1", and stores the
;   numerical representation of the 4 octets in the ipOctets variable.
;       arguments: pointer to the IP address string
;       returns: 0 on success, -1 on failure
;
cStrIP_to_Octets:
    push ebp    ;push ebp ontp the stacl
    mov ebp, esp    ; move the stack pointer into the base pointer

    ; Allocate space for a temporary 3 digit substring variable of the IP
    ; address, used to parse the IP address.
    sub esp, 4  ;subtract 4 bytes from stack pointer

    ; Point esi to the beginning of the string
    mov esi, [ebp+8]    ;move the address of base pointer + 8 into esi

    ; Reset our counter, we'll use this to iterate through the
    ; 3 digits of each octet.
    mov ecx, 0  ;move 0 into ecx

    ; Reset our octet counter, this is to keep track of the 4
    ; octets we need to fill.
    mov edx, 0  ;move zero into edx

    ; Point edi to the beginning of the temporary
    ; IP octet substring
    mov edi, ebp    ;move base pointer into edi
    sub edi, 4  ;subtract 4 from edi

    string_ip_parse_loop:
        ; Read the next character from the IP string
        lodsb   ;Load byte at address DS:ESI into AL
        ; Increment our counter
        inc ecx ;increment ecx counter
```

;continued on next page

```
; If we encounter a dot, process this octet
        cmp al, '.' ; compare al to '.'
        je octet_complete   ;jump if equal to octet_complete
        ; If we encounter a null character, process this
        ; octet.
        cmp al, 0   ;compare al to 0
        je null_byte_encountered    ;jump if equal to null_byte_encountered
        ; If we're already on our third digit,
        ; process this octet.
        cmp ecx, 4  ;compare ecx to 4
        jge octet complete  ;if ecx is greater than 4 jump to octet complete

        ; Otherwise, copy the character to our
        ; temporary octet string.
        stosb   ;Store AL at address ES:EDI

        jmp string_ip_parse_loop ;jump to string_ip_parse_loop

    null byte encountered:
        ; Check to see if we are on the last octet yet
        ; (current octet would be equal to 3)
        cmp edx, 3  ;compare edx to 3
        ; If so, everything is working normally
        je octet_complete   ;jump if equal to octet_complete
        ; Otherwise, this is a malformed IP address,
        ; and we will return -1 for failure
        mov eax, -1 ;move -1 into eax
        jmp malformed ip address exit   ;jump to malformed ip address exit

    octet complete:
        ; Null terminate our temporary octet variable.
        mov al, 0   ;move 0 into al
        stosb   ;Store AL at address ES:EDI

        ; Save our position in the IP address string
        push esi    ;push esi onto the stack
        ; Save our octet counter
        push edx    ;push edx onto the stack
```

```
; Send off our temporary octet string to our cStrtoul
        ; function to turn it into a number.
        mov eax, ebp    ;move base pointer into eax
        sub eax, 4  ;subtract 4 from eax
        push eax    ;push eax onto the stack
        call cStrtoul   ;call cStrtoul routine
        add esp, 4  ;remove 4 bytes from stack pointer

        ; Check if we had any errors converting the string,
        ; if so, go straight to exit (eax will hold error through)
        cmp eax, 0  ;compare eax to zero
        jl malformed ip address exit    ;if less than jump to
malformed_ip_address_exit

        ; Restore our octet counter
        pop edx ;pop edx from stack

        ; Copy the octet data to the current IP octet
        ; in our IP octet array.
        mov edi, ipOctets   ;move ipOctets variable into edi
        add edi, edx    ;add edx to edu
        ; cStrtoul saved the number in eax, so we should
        ; be fine writing al to [edi].
        stosb   ;Store AL at address ES:EDI

        ; Increment our octet counter.
        inc edx ;add 1 to edx

        ; Restore our position in the IP address string
        pop esi ;pop esi from the stack
        ; Reset the position on the temporary octet string
        mov edi, ebp    ;move base pointer into edi
        sub edi, 4  ;subtract 4 from edu
        ; Continue to processing the next octet
        mov ecx, 0  ;move 0 into ecx

        cmp edx, 4  ;compare edx to 4
        jl string ip parse loop ;if less than jump to
string ip parse loop

    ; Return 0 for success
    mov eax, 0  ;move 0 into eax

    malformed_ip_address_exit:
    mov esp, ebp    ;move base pointer into stack pointer
    pop ebp ;pop base pointer
    ret ;return
```

```
;return to main() to validate ip address
```

```
;Checking for errors in IP address
```

```
add esp, 4  ;remove top 4 bytes from the stack
pointer
; Check for errors
cmp eax, 0  ;compare eax to zero
;if less than, jump to "invalid_program_arguments"
jl invalid_program_arguments
```

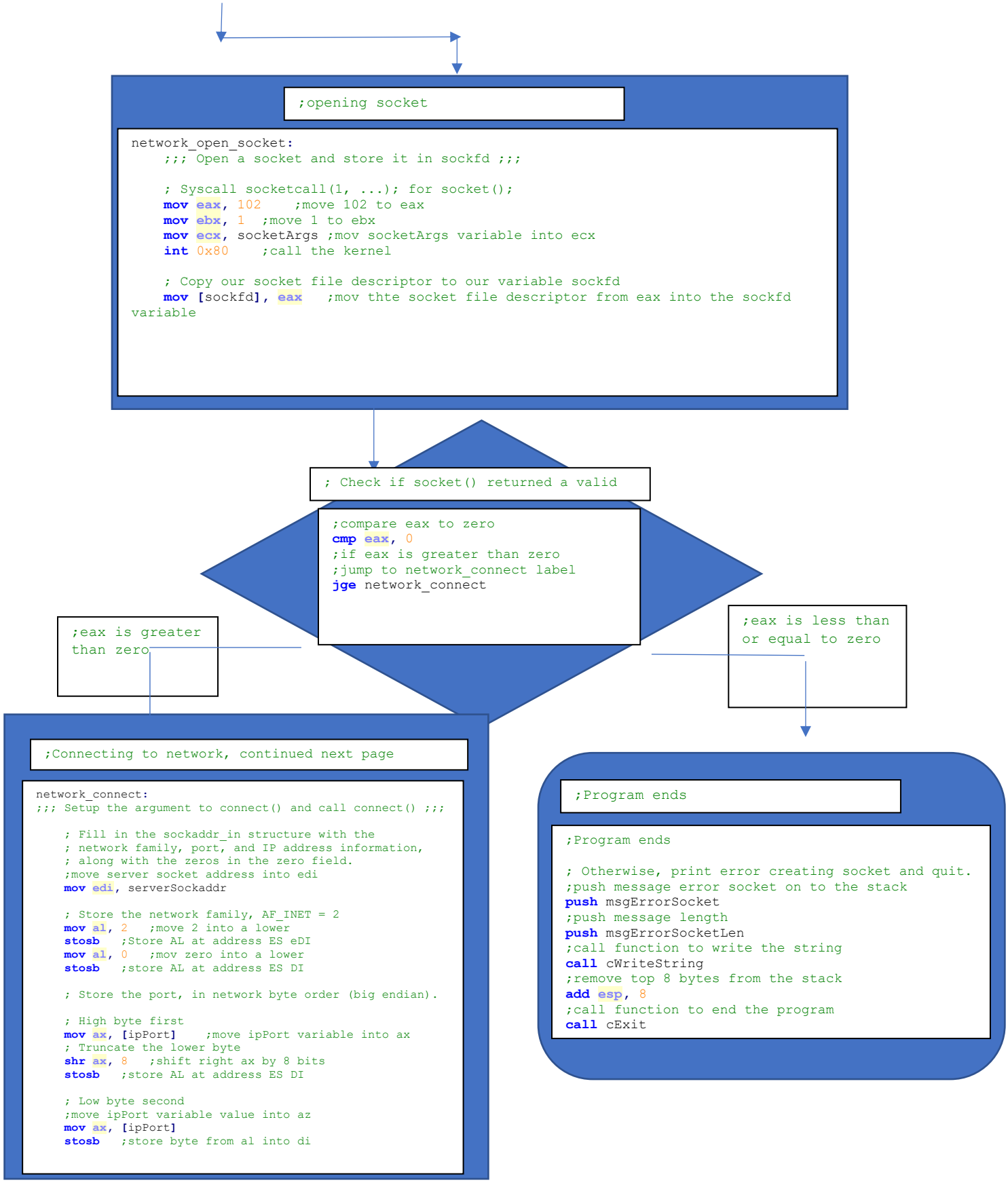
```
;error check
greater than
zero
```

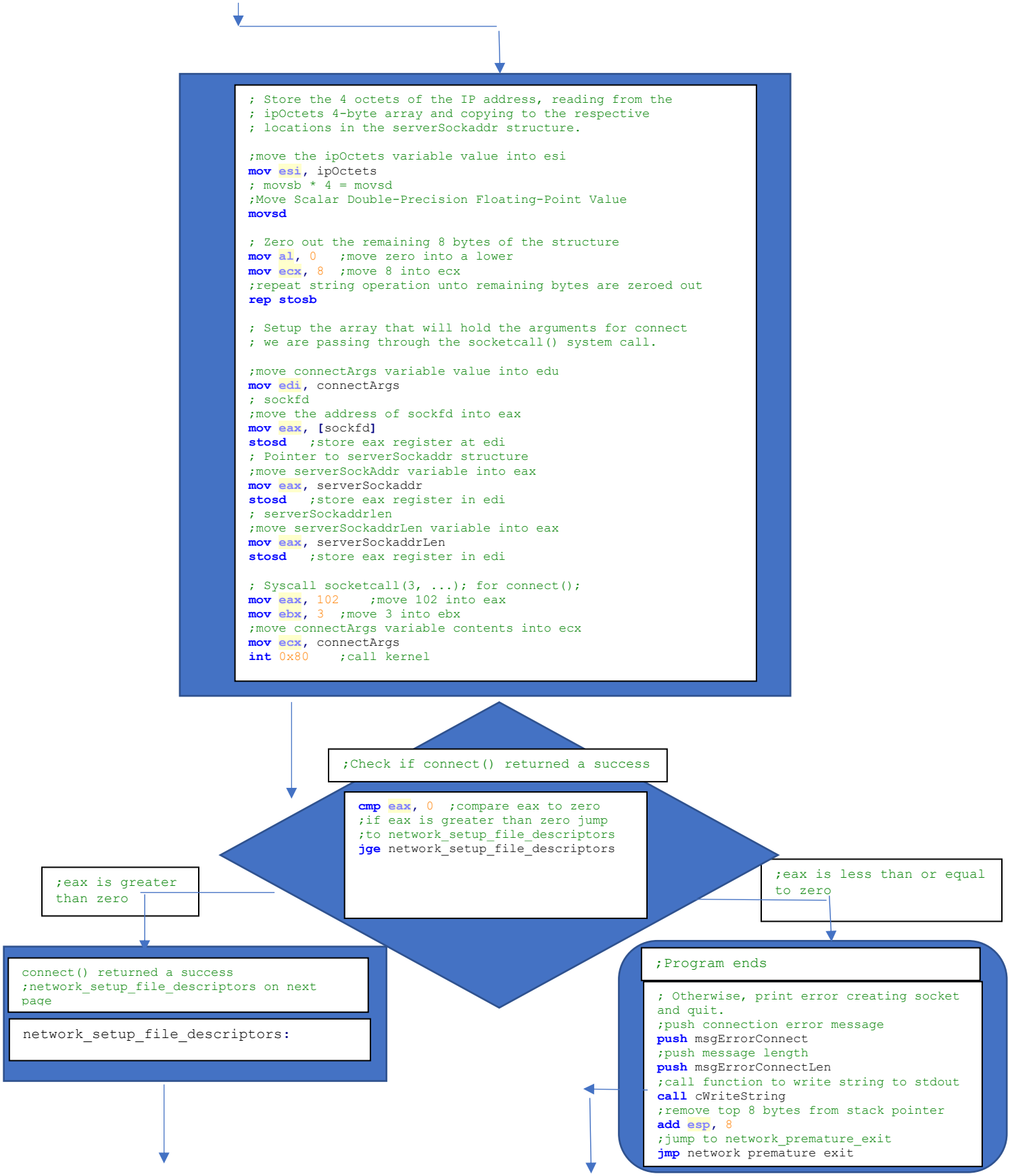
```
;error checking
less than zero
```

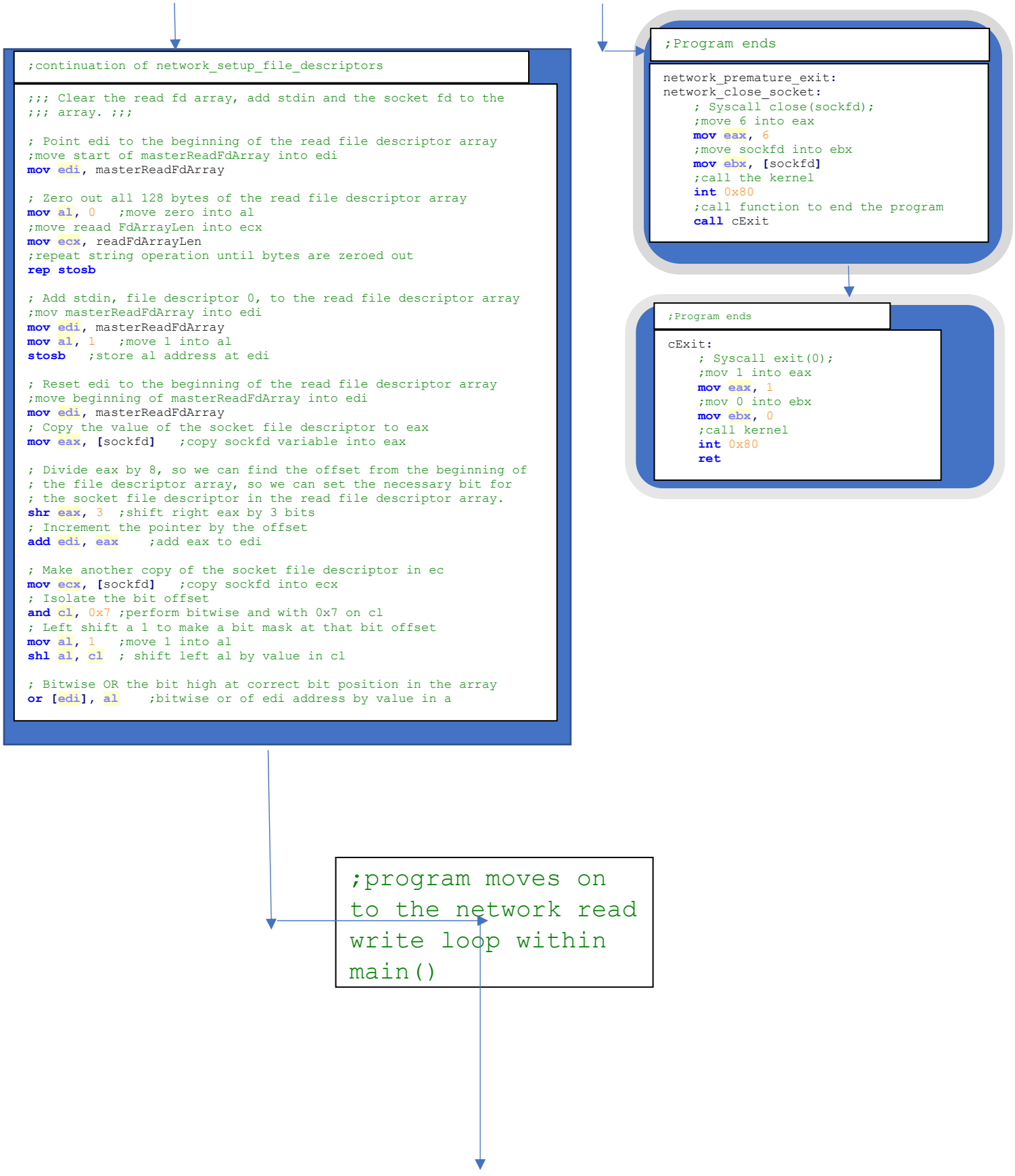
```
; Next on the stack is the port
; Convert the port string to a 16-bit word.
;call function to convert port string to
work
call cStrtoul
;remove top 4 bytes from the extended stack
pointer
add esp, 4
;move the contents of eax into the address
of the ipPort variable
mov [ipPort], eax
```

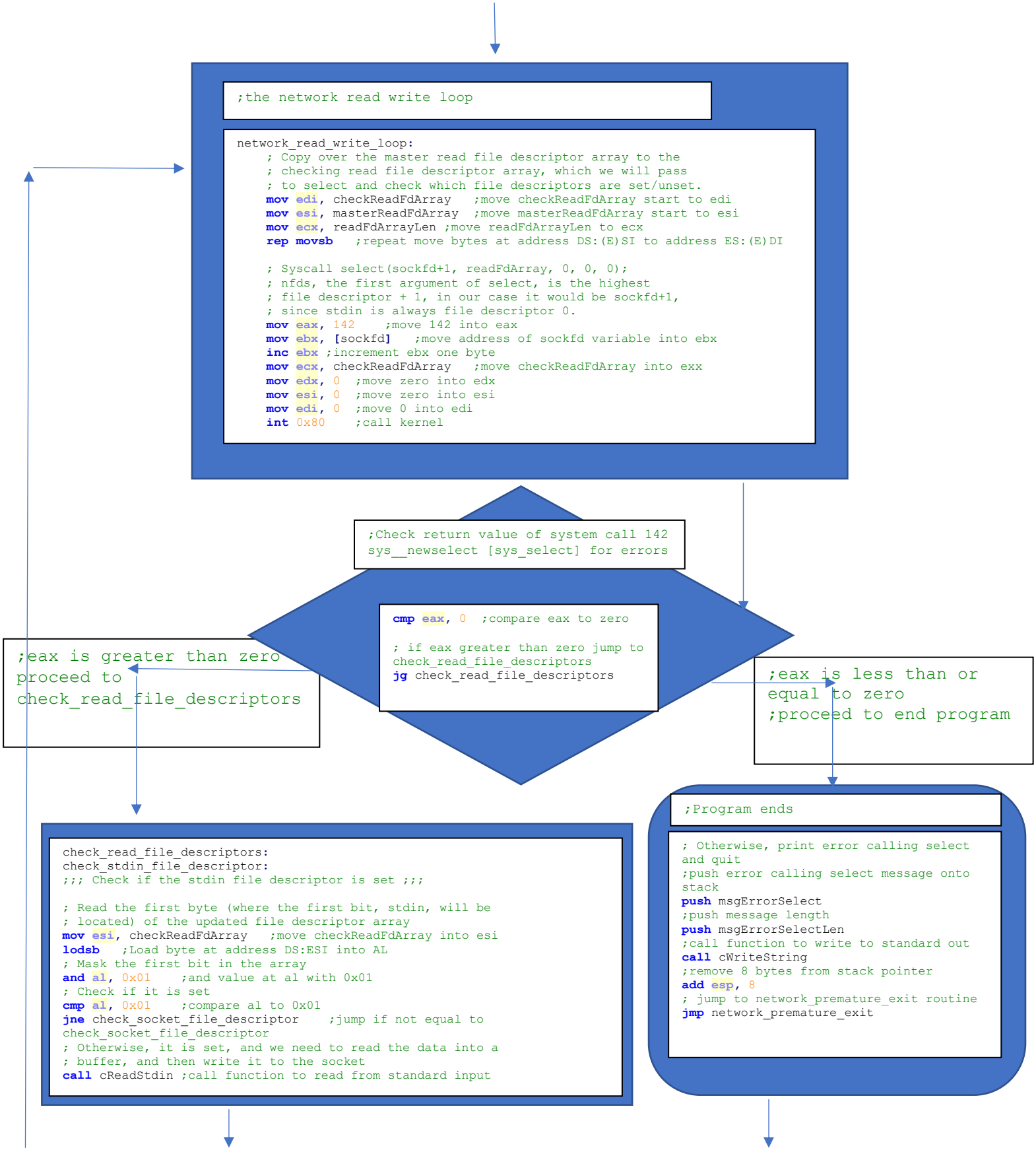
```
;Program ends
```

```
;fall into invalid_program_arguments if eax is not
greater than zero
invalid_program_arguments:
    ;push the invalid arguments message on the stack
    push msgInvalidArguments
    ;push the message length
    push msgInvalidArgumentsLen
    ;call the function to write the string
    call cWriteString
    ;remove top 8 bytes from stack pointer
    add esp, 8
    ;call function to exit the program
    call cExit
```

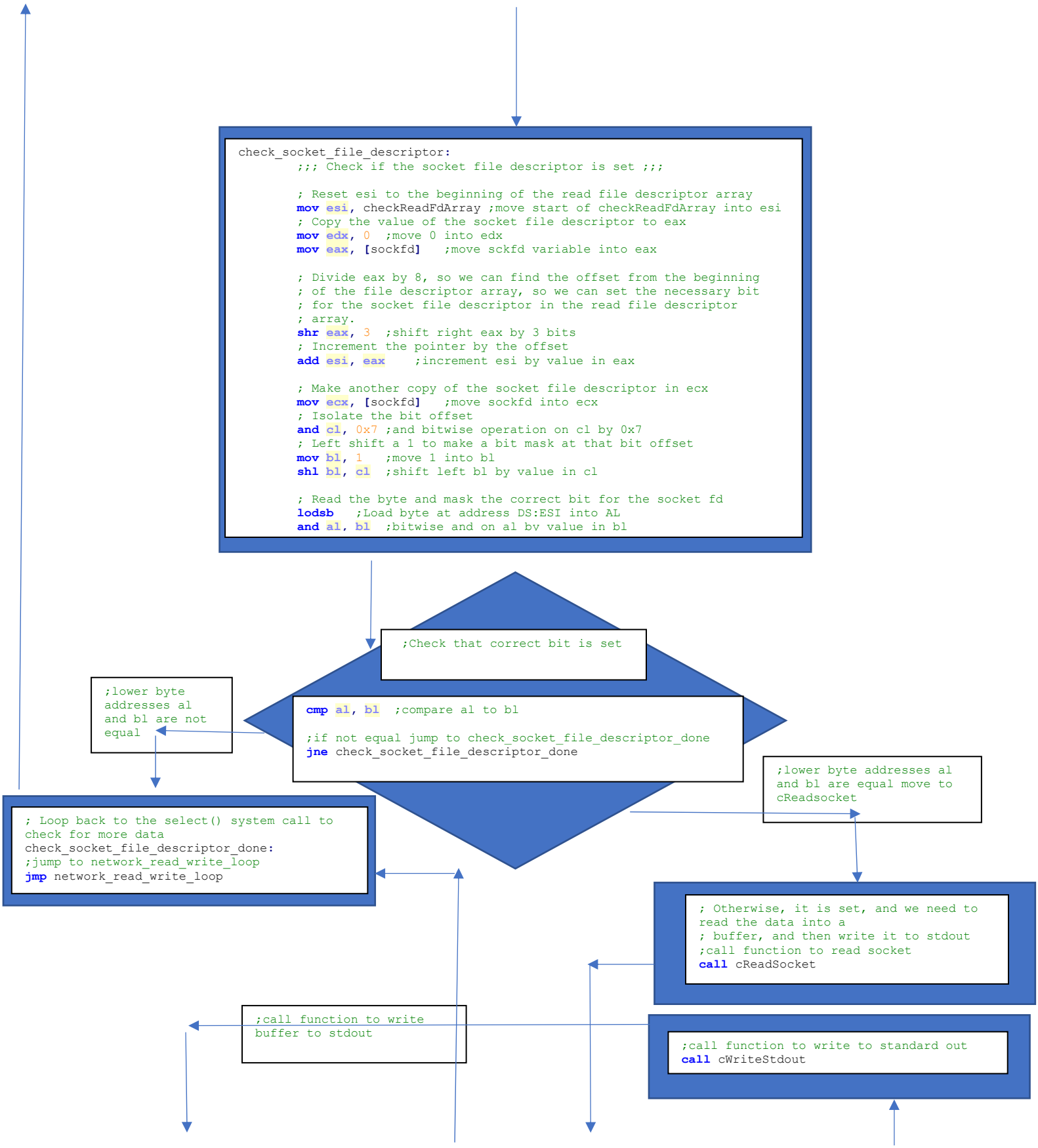

;opening socket
network_open_socket:
;;; Open a socket and store it in sockfd ;;;
; Syscall socketcall(1, ...); for socket();
mov eax, 102 ;move 102 to eax
mov ebx, 1 ;move 1 to ebx
mov ecx, socketArgs ;mov socketArgs variable into ecx
int 0x80 ;call the kernel
; Copy our socket file descriptor to our variable sockfd
mov [sockfd], eax ;mov thte socket file descriptor from eax into the sockfd variable
; Check if socket() returned a valid
;compare eax to zero
cmp eax, 0
;if eax is greater than zero
;jump to network_connect label
jge network_connect
;eax is greater than zero
;eax is less than or equal to zero
;Connecting to network, continued next page
network_connect:
;;; Setup the argument to connect() and call connect() ;;;
; Fill in the sockaddr_in structure with the
; network family, port, and IP address information,
; along with the zeros in the zero field.
;move server socket address into edi
mov edi, serverSockaddr
; Store the network family, AF_INET = 2
mov al, 2 ;move 2 into a lower
stosb ;Store AL at address ES eDI
mov al, 0 ;mov zero into a lower
stosb ;store AL at address ES DI
; Store the port, in network byte order (big endian).
; High byte first
mov ax, [ipPort] ;move ipPort variable into ax
; Truncate the lower byte
shr ax, 8 ;shift right ax by 8 bits
stosb ;store AL at address ES DI
; Low byte second
;move ipPort variable value into az
mov ax, [ipPort]
stosb ;store byte from al into di
;Program ends
;Program ends
; Otherwise, print error creating socket and quit.
;push message error socket on to the stack
push msgErrorSocket
;push message length
push msgErrorSocketLen
;call function to write the string
call cWriteString
;remove top 8 bytes from the stack
add esp, 8
;call function to end the program
call cExit

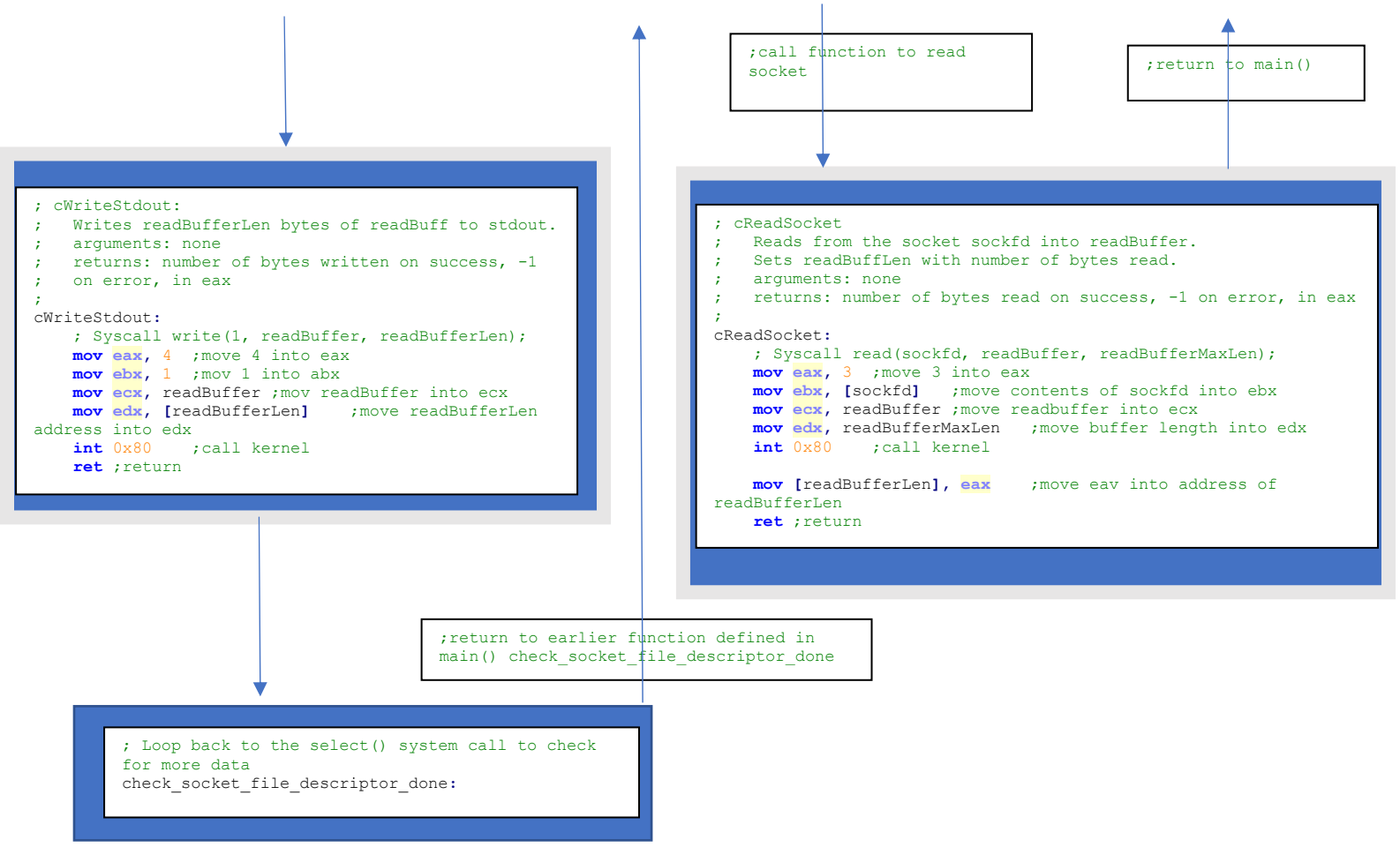

; Store the 4 octets of the IP address, reading from the
; ipOctets 4-byte array and copying to the respective
; locations in the serverSockaddr structure.
;move the ipOctets variable value into esi
mov esi, ipOctets
; movsb * 4 = movsd
;Move Scalar Double-Precision Floating-Point Value
movsd
; Zero out the remaining 8 bytes of the structure
mov al, 0 ;move zero into a lower
mov ecx, 8 ;move 8 into ecx
;repeat string operation unto remaining bytes are zeroed out
rep stosb
; Setup the array that will hold the arguments for connect
; we are passing through the socketcall() system call.
;move connectArgs variable value into edu
mov edi, connectArgs
; sockfd
;move the address of sockfd into eax
mov eax, [sockfd]
stosd ;store eax register at edi
; Pointer to serverSockaddr structure
;move serverSockAddr variable into eax
mov eax, serverSockaddr
stosd ;store eax register in edi
; serverSockaddrlen
;move serverSockaddrLen variable into eax
mov eax, serverSockaddrLen
stosd ;store eax register in edi
; Syscall socketcall(3, ...); for connect();
mov eax, 102 ;move 102 into eax
mov ebx, 3 ;move 3 into ebx
;move connectArgs variable contents into ecx
mov ecx, connectArgs
int 0x80 ;call kernel
;Check if connect() returned a success
cmp eax, 0 ;compare eax to zero
;if eax is greater than zero jump
;to network_setup_file_descriptors
jge network_setup_file_descriptors
;eax is greater than zero
;eax is less than or equal to zero
connect() returned a success
;network_setup_file_descriptors on next page
network_setup_file_descriptors:
;Program ends
; Otherwise, print error creating socket and quit.
;push connection error message
push msgErrorConnect
;push message length
push msgErrorConnectLen
;call function to write string to stdout
call cWriteString
;remove top 8 bytes from stack pointer
add esp, 8
;jump to network_premature_exit
jmp network premature exit


;continuation of network_setup_file_descriptors
;;; Clear the read fd array, add stdin and the socket fd to the
;;; array. ;;;
; Point edi to the beginning of the read file descriptor array
;move start of masterReadFdArray into edi
mov edi, masterReadFdArray
; Zero out all 128 bytes of the read file descriptor array
mov al, 0 ;move zero into al
;move reaad FdArrayLen into ecx
mov ecx, readFdArrayLen
;repeat string operation until bytes are zeroed out
rep stosb
; Add stdin, file descriptor 0, to the read file descriptor array
;mov masterReadFdArray into edi
mov edi, masterReadFdArray
mov al, 1 ;move 1 into al
stosb ;store al address at edi
; Reset edi to the beginning of the read file descriptor array
;move beginning of masterReadFdArray into edi
mov edi, masterReadFdArray
; Copy the value of the socket file descriptor to eax
mov eax, [sockfd] ;copy sockfd variable into eax
; Divide eax by 8, so we can find the offset from the beginning of
; the file descriptor array, so we can set the necessary bit for
; the socket file descriptor in the read file descriptor array.
shr eax, 3 ;shift right eax by 3 bits
; Increment the pointer by the offset
add edi, eax ;add eax to edi
; Make another copy of the socket file descriptor in ec
mov ecx, [sockfd] ;copy sockfd into ecx
; Isolate the bit offset
and cl, 0x7 ;perform bitwise and with 0x7 on cl
; Left shift a 1 to make a bit mask at that bit offset
mov al, 1 ;move 1 into al
shl al, cl ; shift left al by value in cl
; Bitwise OR the bit high at correct bit position in the array
or [edi], al ;bitwise or of edi address by value in a
;Program ends
network_premature_exit:
network_close_socket:
; Syscall close(sockfd);
;move 6 into eax
mov eax, 6
;move sockfd into ebx
mov ebx, [sockfd]
;call the kernel
int 0x80
;call function to end the program
call cExit
;Program ends
cExit:
; Syscall exit(0);
;mov 1 into eax
mov eax, 1
;mov 0 into ebx
mov ebx, 0
;call kernel
int 0x80
ret
;program moves on to the network read write loop within main()


;the network read write loop
network_read_write_loop:
; Copy over the master read file descriptor array to the
; checking read file descriptor array, which we will pass
; to select and check which file descriptors are set/unset.
mov edi, checkReadFdArray ;move checkReadFdArray start to edi
mov esi, masterReadFdArray ;move masterReadFdArray start to esi
mov ecx, readFdArrayLen ;move readFdArrayLen to ecx
rep movsb ;repeat move bytes at address DS:(E)SI to address ES:(E)DI
; Syscall select(sockfd+1, readFdArray, 0, 0, 0);
; nfds, the first argument of select, is the highest
; file descriptor + 1, in our case it would be sockfd+1,
; since stdin is always file descriptor 0.
mov eax, 142 ;move 142 into eax
mov ebx, [sockfd] ;move address of sockfd variable into ebx
inc ebx ;increment ebx one byte
mov ecx, checkReadFdArray ;move checkReadFdArray into exx
mov edx, 0 ;move zero into edx
mov esi, 0 ;move zero into esi
mov edi, 0 ;move 0 into edi
int 0x80 ;call kernel
;Check return value of system call 142
sys__newselect [sys_select] for errors
cmp eax, 0 ;compare eax to zero
; if eax greater than zero jump to
check_read_file_descriptors
jg check_read_file_descriptors
;eax is greater than zero
proceed to
check_read_file_descriptors
;eax is less than or
equal to zero
;proceed to end program
check_read_file_descriptors:
check_stdin_file_descriptor:
;;; Check if the stdin file descriptor is set ;;;
; Read the first byte (where the first bit, stdin, will be
; located) of the updated file descriptor array
mov esi, checkReadFdArray ;move checkReadFdArray into esi
lodsb ;Load byte at address DS:ESI into AL
; Mask the first bit in the array
and al, 0x01 ;and value at al with 0x01
; Check if it is set
cmp al, 0x01 ;compare al to 0x01
jne check_socket_file_descriptor ;jump if not equal to
check_socket_file_descriptor
; Otherwise, it is set, and we need to read the data into a
; buffer, and then write it to the socket
call cReadStdin ;call function to read from standard input
;Program ends
; Otherwise, print error calling select
and quit
;push error calling select message onto
stack
push msgErrorSelect
;push message length
push msgErrorSelectLen
;call function to write to standard out
call cWriteString
;remove 8 bytes from stack pointer
add esp, 8
; jump to network_premature_exit routine
jmp network_premature_exit


;jumping to cReadStdin function
; cReadStdin
; Reads from stdin into readBuffer.
; Sets readBuffLen with number of bytes read.
; arguments: none
; returns: number of bytes read on success,
; -1 on error, in eax
cReadStdin:
; Syscall read(0, readBuffer, readBufferMaxLen);
mov eax, 3 ;move 3 into eax
mov ebx, 0 ;move 0 into ebx
mov ecx, readBuffer ;move readBuffer into ecx
mov edx, readBufferMaxLen ;move readBufferMaxLen into edx
int 0x80 ;call the kernel
;move eax into readBufferLen variable address
mov [readBufferLen], eax
ret ;return
;back to main() where call cwriteSocket is called
call cWriteSocket ;call function to write socket
; cWriteSocket
; Writes readBufferLen bytes of readBuff to the socket sockfd.
; arguments: none
; returns: number of bytes written on success, -1 on error, in eax
;
cWriteSocket:
; Syscall write(sockfd, readBuff, readBuffLen);
mov eax, 4 ;move 4 into eax
mov ebx, [sockfd] ;move sockfd address into ebx
mov ecx, readBuffer ;move buffer into ecx
mov edx, [readBufferLen] ;move address of buffer length into edx
int 0x80 ;call kernel
ret ;return
;Program ends
network_premature_exit:
network_close_socket:
; Syscall close(sockfd);
;move 6 into eax
mov eax, 6
;move sockfd into ebx
mov ebx, [sockfd]
;call the kernel
int 0x80
;call function to end the program
call cExit
cExit:
; Syscall exit(0);
;mov 1 into eax
mov eax, 1
;mov 0 into ebx
mov ebx, 0
;call kernel
int 0x80
ret
;return to main()
;Checking the socket file descriptor
;and ensuring file descriptor is set
;Continued on next page


check_socket_file_descriptor:
;;; Check if the socket file descriptor is set ;;;
; Reset esi to the beginning of the read file descriptor array
mov esi, checkReadFdArray ;move start of checkReadFdArray into esi
; Copy the value of the socket file descriptor to eax
mov edx, 0 ;move 0 into edx
mov eax, [sockfd] ;move sckfd variable into eax
; Divide eax by 8, so we can find the offset from the beginning
; of the file descriptor array, so we can set the necessary bit
; for the socket file descriptor in the read file descriptor
; array.
shr eax, 3 ;shift right eax by 3 bits
; Increment the pointer by the offset
add esi, eax ;increment esi by value in eax
; Make another copy of the socket file descriptor in ecx
mov ecx, [sockfd] ;move sockfd into ecx
; Isolate the bit offset
and cl, 0x7 ;and bitwise operation on cl by 0x7
; Left shift a 1 to make a bit mask at that bit offset
mov bl, 1 ;move 1 into bl
shl bl, cl ;shift left bl by value in cl
; Read the byte and mask the correct bit for the socket fd
lodsb ;Load byte at address DS:ESI into AL
and al, bl ;bitwise and on al by value in bl
;Check that correct bit is set
cmp al, bl ;compare al to bl
;if not equal jump to check_socket_file_descriptor_done
jne check_socket_file_descriptor_done
;lower byte addresses al and bl are not equal
; Loop back to the select() system call to check for more data
check_socket_file_descriptor_done:
;jump to network_read_write_loop
jmp network_read_write_loop
;lower byte addresses al and bl are equal move to cReadsocket
; Otherwise, it is set, and we need to read the data into a
; buffer, and then write it to stdout
;call function to read socket
call cReadSocket
;call function to write to standard out
call cWriteStdout
;call function to write buffer to stdout


;call function to read socket
;return to main()
; cWriteStdout:
; Writes readBufferLen bytes of readBuff to stdout.
; arguments: none
; returns: number of bytes written on success, -1
; on error, in eax
;
cWriteStdout:
; Syscall write(1, readBuffer, readBufferLen);
mov eax, 4 ;move 4 into eax
mov ebx, 1 ;mov 1 into abx
mov ecx, readBuffer ;mov readBuffer into ecx
mov edx, [readBufferLen] ;move readBufferLen address into edx
int 0x80 ;call kernel
ret ;return
; cReadSocket
; Reads from the socket sockfd into readBuffer.
; Sets readBuffLen with number of bytes read.
; arguments: none
; returns: number of bytes read on success, -1 on error, in eax
;
cReadSocket:
; Syscall read(sockfd, readBuffer, readBufferMaxLen);
mov eax, 3 ;move 3 into eax
mov ebx, [sockfd] ;move contents of sockfd into ebx
mov ecx, readBuffer ;move readbuffer into ecx
mov edx, readBufferMaxLen ;move buffer length into edx
int 0x80 ;call kernel
mov [readBufferLen], eax ;move eav into address of readBufferLen
ret ;return
;return to earlier function defined in main() check_socket_file_descriptor_done
; Loop back to the select() system call to check for more data
check_socket_file_descriptor_done: