100507515

Implementation Log

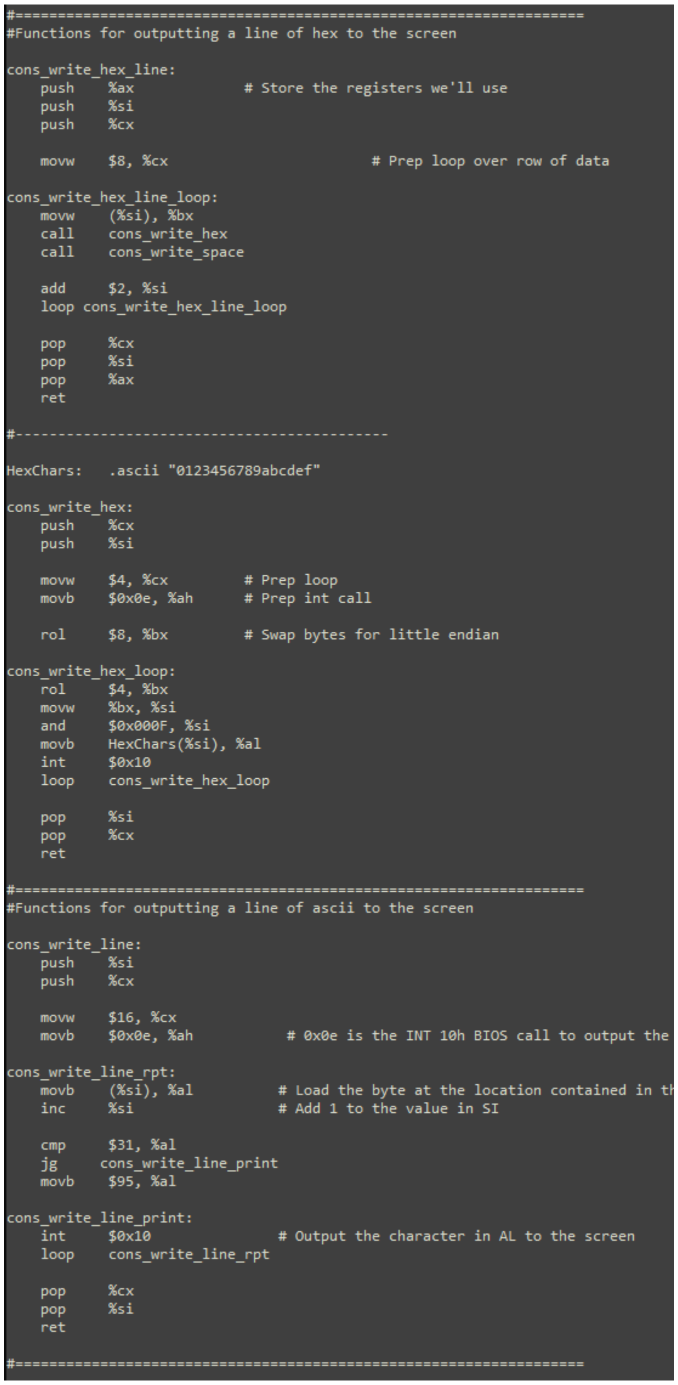
After finishing the implementation, I realized that early on in development, I mistook the terms sector and block when looking them up to mean similar but slightly different things. Throughout my code, the term block is often referred to as the initial starting sector when reading from disk. I've updated all of the strings and visible stuff however a lot of the function names, variables and comments may reference the term block instead of sector.

27/10/23

Started on Assignment. Started with a clean week 2 solution as a starting point. Spent the 2 workshop hours initially experimenting with the int 16h call, learning what it did and creating a simple “get and display character” function but didn’t really use in the end. Then spent the rest of the time planning and then starting the implementation of a display disk sector function to just display a certain number of lines in memory. I also tried moving the bootasm2 closer to bootasm1 in memory and settled on address 0x8000. I think I could move it closer to address 0x7E00 which is the next sector, but I didn’t try.

01/11/23

Spent a couple of hours and finished display disk sector function which looked a bit like as followed.



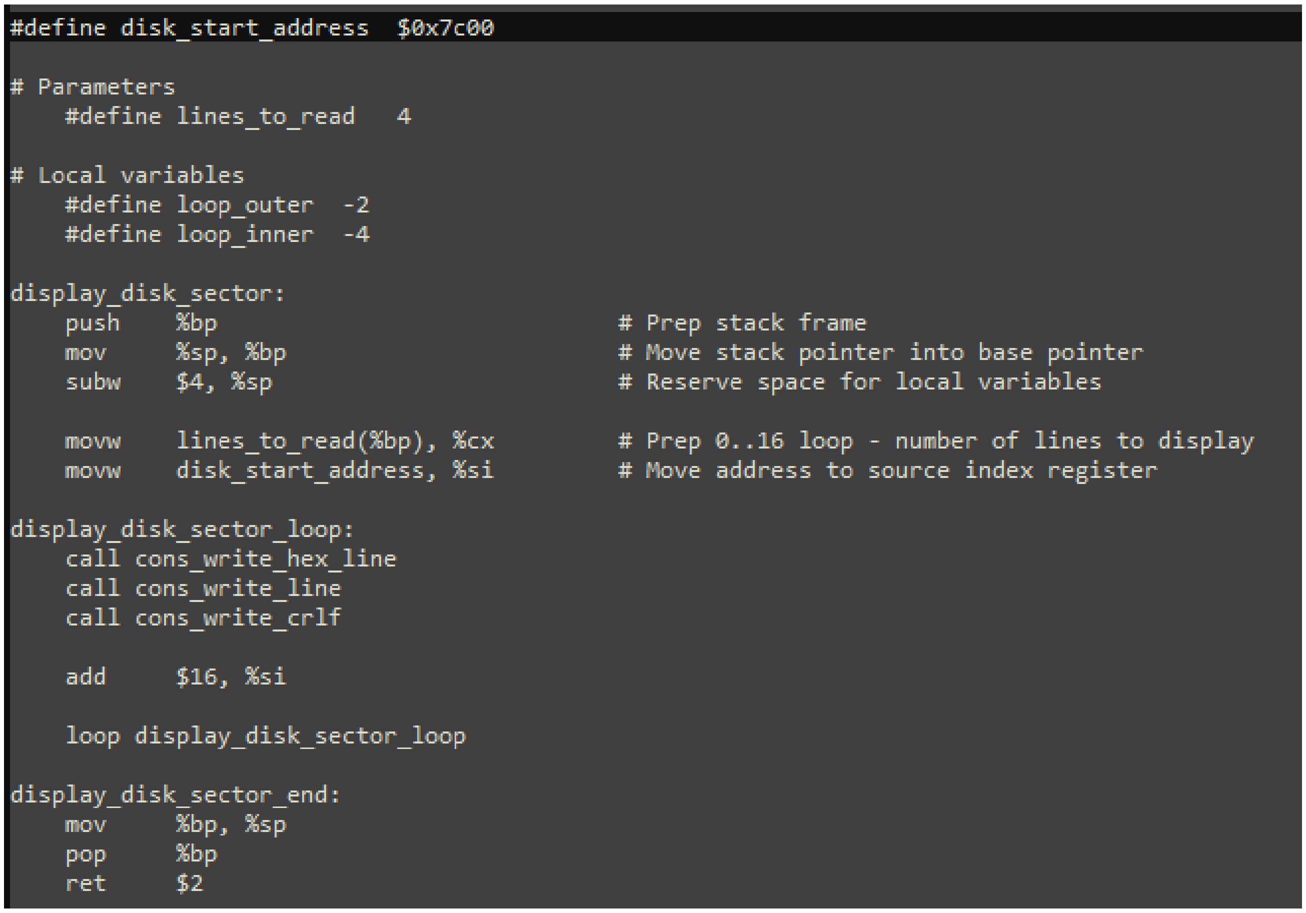
```
#==================================================================
#Functions for outputting a line of hex to the screen

cons_write_hex_line:
    push    %ax             # Store the registers we'll use
    push    %si
    push    %cx

    movw    $8, %cx                        # Prep loop over row of data

cons_write_hex_line_loop:
    movw    (%si), %bx
    call    cons_write_hex
    call    cons_write_space

    add     $2, %si
    loop cons_write_hex_line_loop

    pop     %cx
    pop     %si
    pop     %ax
    ret

#---------------------------------------------

HexChars:   .ascii "0123456789abcdef"

cons_write_hex:
    push    %cx
    push    %si

    movw    $4, %cx         # Prep loop
    movb    $0x0e, %ah      # Prep int call

    rol     $8, %bx         # Swap bytes for little endian

cons_write_hex_loop:
    rol     $4, %bx
    movw    %bx, %si
    and     $0x000F, %si
    movb    HexChars(%si), %al
    int     $0x10
    loop    cons_write_hex_loop

    pop     %si
    pop     %cx
    ret

#==================================================================
#Functions for outputting a line of ascii to the screen

cons_write_line:
    push    %si
    push    %cx

    movw    $16, %cx
    movb    $0x0e, %ah           # 0x0e is the INT 10h BIOS call to output the

cons_write_line_rpt:
    movb    (%si), %al          # Load the byte at the location contained in th
    inc     %si                 # Add 1 to the value in SI

    cmp     $31, %al
    jg     cons_write_line_print
    movb    $95, %al

cons_write_line_print:
    int     $0x10               # Output the character in AL to the screen
    loop    cons_write_line_rpt

    pop     %cx
    pop     %si
    ret

#==================================================================
```

Essentially just iterates over an address in memory which I set to the 0x7c00 as I can quite easily check if this if correct or not. I just increase si by 16 bytes every loop for a given number of times and call a function to display the hex and then the ascii.

The hex function is a modified version of the provided cons_write_hex which displays 16 bytes worth of hex in blocks of 2 with spaces in between each block. I initially added a space to the end of HexChars which I thought could be used to display a space between the blocks however couldn’t initially get it to work and opted to create a designated cons_write_space function which I figured I would use later in the code as well anyway.

The cons_write_line, which I need to come up with a better name for, is based on the cons_writeline function but instead of iterating over and checking for the string termination, we just iterate 16 times but we also make sure the value to print is greater than 31.

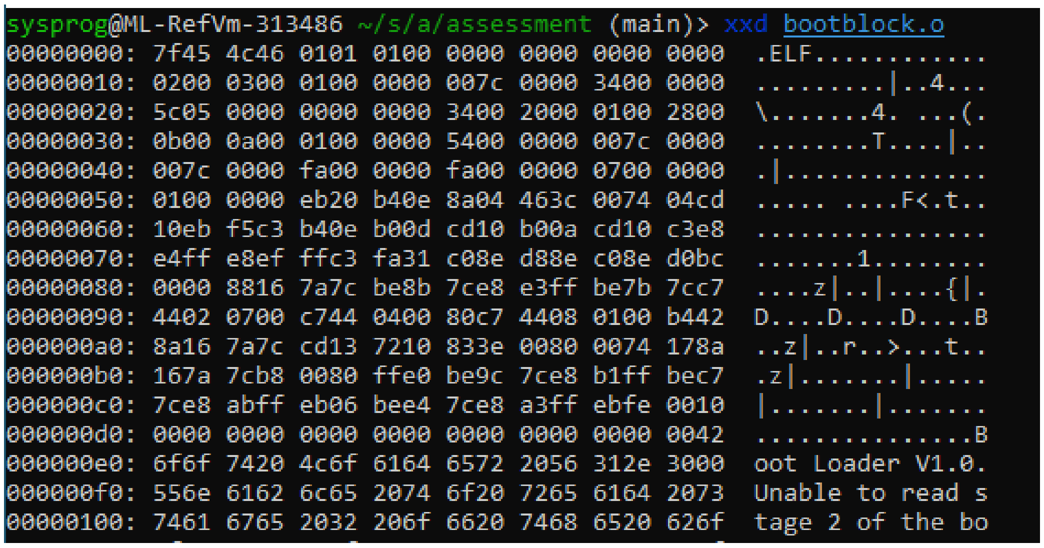
I had some issues getting this to work initially, mainly with accidentally overwriting registers, especially si however once I checked everything and added additionaly push and pop instructions to make sure I wasn’t overwriting anything, everything seemed to work however I couldn’t easly check if the hex was working.



```
#define disk_start_address  $0x7c00

# Parameters
    #define lines_to_read   4

# Local variables
    #define loop_outer  -2
    #define loop_inner  -4

display_disk_sector:
    push    %bp                             # Prep stack frame
    mov     %sp, %bp                        # Move stack pointer into base pointer
    subw    $4, %sp                         # Reserve space for local variables

    movw    lines_to_read(%bp), %cx         # Prep 0..16 loop - number of lines to display
    movw    disk_start_address, %si         # Move address to source index register

display_disk_sector_loop:
    call cons_write_hex_line
    call cons_write_line
    call cons_write_crlf

    add     $16, %si

    loop display_disk_sector_loop

display_disk_sector_end:
    mov     %bp, %sp
    pop     %bp
    ret     $2
```

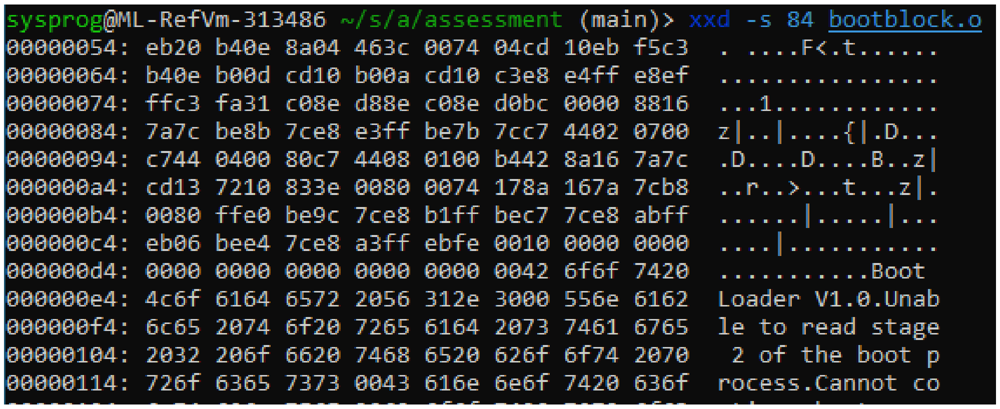
03/11/23

Spent an hour at the workshop finishing the previous function. I got recommended to use an existing hex reader which I could use to find out if my work was reading correctly. I looked into it and found one called xxd which I could install onto the linux subsystem. I used this to read the produced object file which I could compare against what I had loaded into memory which looked like this:



```
sysprog@ML-RefVm-313486 ~/s/a/assessment (main)> xxd bootblock.o
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 0300 0100 0000 007c 0000 3400 0000  .........|..4...
00000020: 5c05 0000 0000 0000 3400 2000 0100 2800  \.......4. ...(.
00000030: 0b00 0a00 0100 0000 5400 0000 007c 0000  ........T....|..
00000040: 007c 0000 fa00 0000 fa00 0000 0700 0000  .|..............
00000050: 0100 0000 eb20 b40e 8a04 463c 0074 04cd  ..... ....F<.t..
00000060: 10eb f5c3 b40e b00d cd10 b00a cd10 c3e8  ................
00000070: e4ff e8ef ffc3 fa31 c08e d88e c08e d0bc  .......1........
00000080: 0000 8816 7a7c be8b 7ce8 e3ff be7b 7cc7  ....z|..|....{|.
00000090: 4402 0700 c744 0400 80c7 4408 0100 b442  D....D....D....B
000000a0: 8a16 7a7c cd13 7210 833e 0080 0074 178a  ..z|..r..>...t..
000000b0: 167a 7cb8 0080 ffe0 be9c 7ce8 b1ff bec7  .z|.......|.....
000000c0: 7ce8 abff eb06 bee4 7ce8 a3ff ebfe 0010  |.......|.......
000000d0: 0000 0000 0000 0000 0000 0000 0000 0042  ...............B
000000e0: 6f6f 7420 4c6f 6164 6572 2056 312e 3000  oot Loader V1.0.
000000f0: 556e 6162 6c65 2074 6f20 7265 6164 2073  Unable to read s
00000100: 7461 6765 2032 206f 6620 7468 6520 626f  tage 2 of the bo
```

This didn’t line up with what I had as I wasn’t expecting an offset. After some experimenting I lined it up like so:



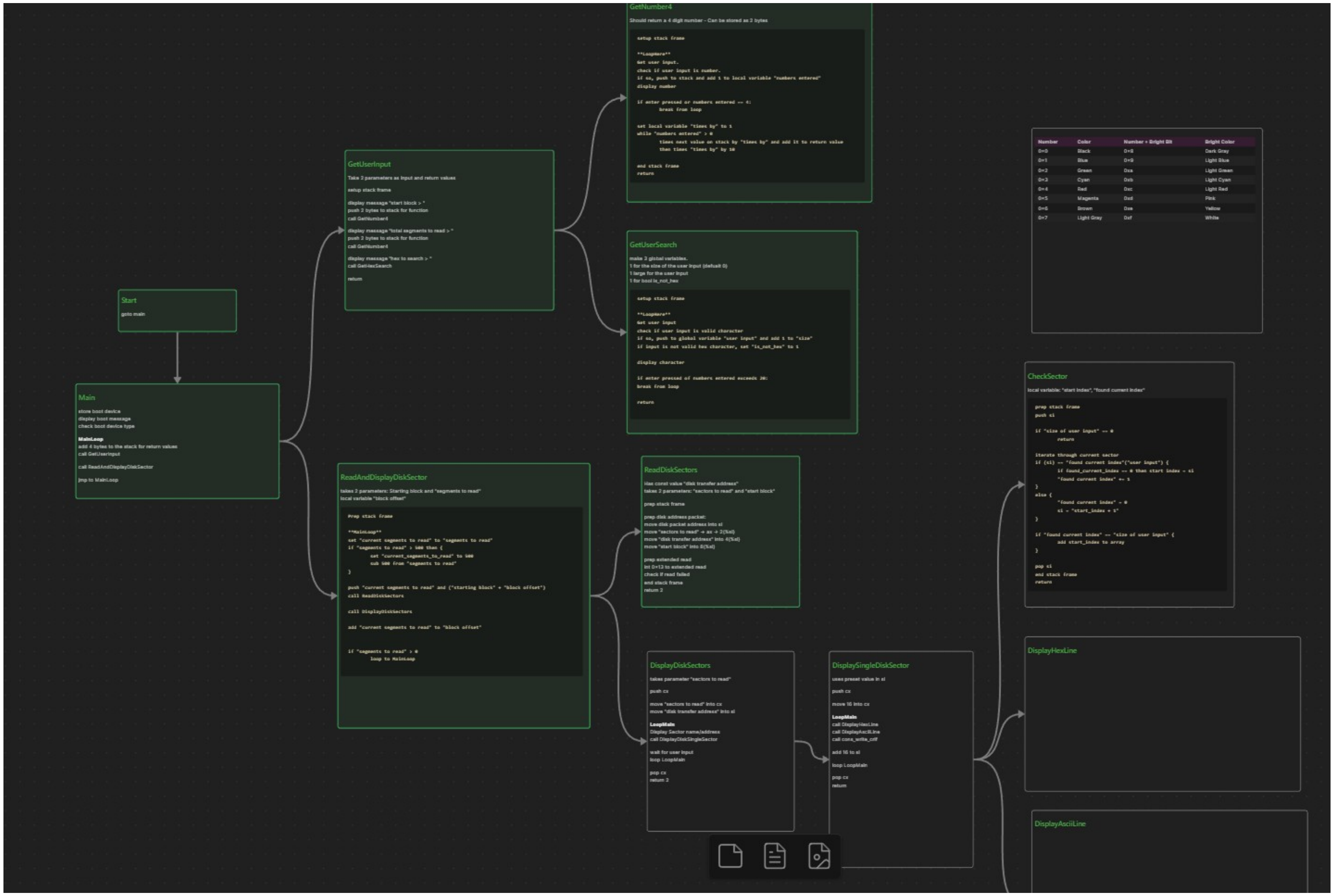
```
sysprog@ML-RefVm-313486 ~/s/a/assessment (main)> xxd -s 84 bootblock.o
00000054: eb20 b40e 8a04 463c 0074 04cd 10eb f5c3  . ....F<.t......
00000064: b40e b00d cd10 b00a cd10 c3e8 e4ff e8ef  ................
00000074: ffc3 fa31 c08e d88e c08e d0bc 0000 8816  ...1............
00000084: 7a7c be8b 7ce8 e3ff be7b 7cc7 4402 0700  z|..|....{|.D...
00000094: c744 0400 80c7 4408 0100 b442 8a16 7a7c  .D....D....B..z|
000000a4: cd13 7210 833e 0080 0074 178a 167a 7cb8  ..r..>...t...z|.
000000b4: 0080 ffe0 be9c 7ce8 b1ff bec7 7ce8 abff  ......|.....|...
000000c4: eb06 bee4 7ce8 a3ff ebfe 0010 0000 0000  ....|...........
000000d4: 0000 0000 0000 0000 0000 0042 6f6f 7420  ...........Boot
000000e4: 4c6f 6164 6572 2056 312e 3000 556e 6162  Loader V1.0.Unab
000000f4: 6c65 2074 6f20 7265 6164 2073 7461 6765  le to read stage
00000104: 2032 206f 6620 7468 6520 626f 6f74 2070   2 of the boot p
00000114: 726f 6365 7373 0043 616e 6e6f 7420 636f  rocess.Cannot co
```

Comparing this to my code, the ascii was correct but the hex was wrong. It took me a little too long to realize that the bytes were flipped around (i.e. instead of eb20, I had 20eb), I assume due to the endianness. I added a line in cons_write_hex (visible in first screenshot) that rolled the bytes 8 bits so that they were flipped which solved it.

07/11/23 – 09/11/23

Spent some time (I don’t know how much exactly) over these days trying to come up with some pseudocode for the overall program as I was losing focus on what I was doing. I used a notetaking app called Obsidian which had a canvas/graph like mode. I’ve tried to show screenshots below of it as well as possible.

Overall image



Start

goto main

Main

store boot device
display boot message
check boot device type

**MainLoop**
add 4 bytes to the stack for return values
call GetUserInput

call ReadAndDisplayDiskSector

jmp to MainLoop

GetUserInput

Take 2 parameters as input and return values

setup stack frame

display message "start block > "
push 2 bytes to stack for function
call GetNumber4

display message "total segments to read > "
push 2 bytes to stack for function
call GetNumber4

display message "hex to search > "
call GetHexSearch

return

GetNumber4

Should return a 4 digit number - Can be stored as 2 bytes

```
setup stack frame

**LoopHere**
Get user input.
check if user input is number.
if so, push to stack and add 1 to local variable "numbers entered"
display number

if enter pressed or numbers entered == 4:
        break from loop

set local variable "times by" to 1
while "numbers entered" > 0
        times next value on stack by "times by" and add it to return value
        then times "times by" by 10


end stack frame
return
```

GetUserSearch

make 3 global variables.
1 for the size of the user input (default 0)
1 large for the user input
1 for bool is_not_hex

```
setup stack frame

**LoopHere**
Get user input
check if user input is valid character
if so, push to global variable "user input" and add 1 to "size"
if input is not valid hex character, set "is_not_hex" to 1

display character

if enter pressed of numbers entered exceeds 20:
break from loop

return
```

ReadAndDisplayDiskSector

takes 2 parameters: Starting block and "segments to read"
local variable "block offset"

```
Prep stack frame

**MainLoop**
set "current segments to read" to "segments to read"
if "segments to read" > 500 then {
        set "current_segments_to_read" to 500
        sub 500 from "segments to read"
}

push "current segments to read" and ("starting block" + "block offset")
call ReadDiskSectors

call DisplayDiskSectors

add "current segments to read" to "block offset"


if "segments to read" > 0
        loop to MainLoop
```

ReadDiskSectors

Has const value "disk transfer address"
takes 2 parameters: "sectors to read" and "start block"

prep stack frame

prep disk address packet:
move disk packet address into si
move "sectors to read" → ax → 2(%si)
move "disk transfer address" into 4(%si)
move "start block" into 8(%si)

prep extended read
int 0×13 to extended read
check if read failed
end stack frame
return 2

DisplayDiskSectors

takes parameter "sectors to read"

push cx

move "sectors to read" into cx
move "disk transfer address" into si

**LoopMain**
Display Sector name/address
call DisplayDiskSingleSector

wait for user input
loop LoopMain

pop cx
return 2

DisplaySingleDiskSector

uses preset value in si

push cx

move 16 into cx

**LoopMain**
call DisplayHexLine
call DisplayAsciiLine
call cons_write_crlf

add 16 to si

loop LoopMain

pop cx
return

| Number | Color | Number + Bright Bit | Bright Color |
|---|---|---|---|
| 0×0 | Black | 0×8 | Dark Gray |
| 0×1 | Blue | 0×9 | Light Blue |
| 0×2 | Green | 0xa | Light Green |
| 0×3 | Cyan | 0xb | Light Cyan |
| 0×4 | Red | 0xc | Light Red |
| 0×5 | Magenta | 0xd | Pink |
| 0×6 | Brown | 0xe | Yellow |
| 0×7 | Light Gray | 0xf | White |

CheckSector

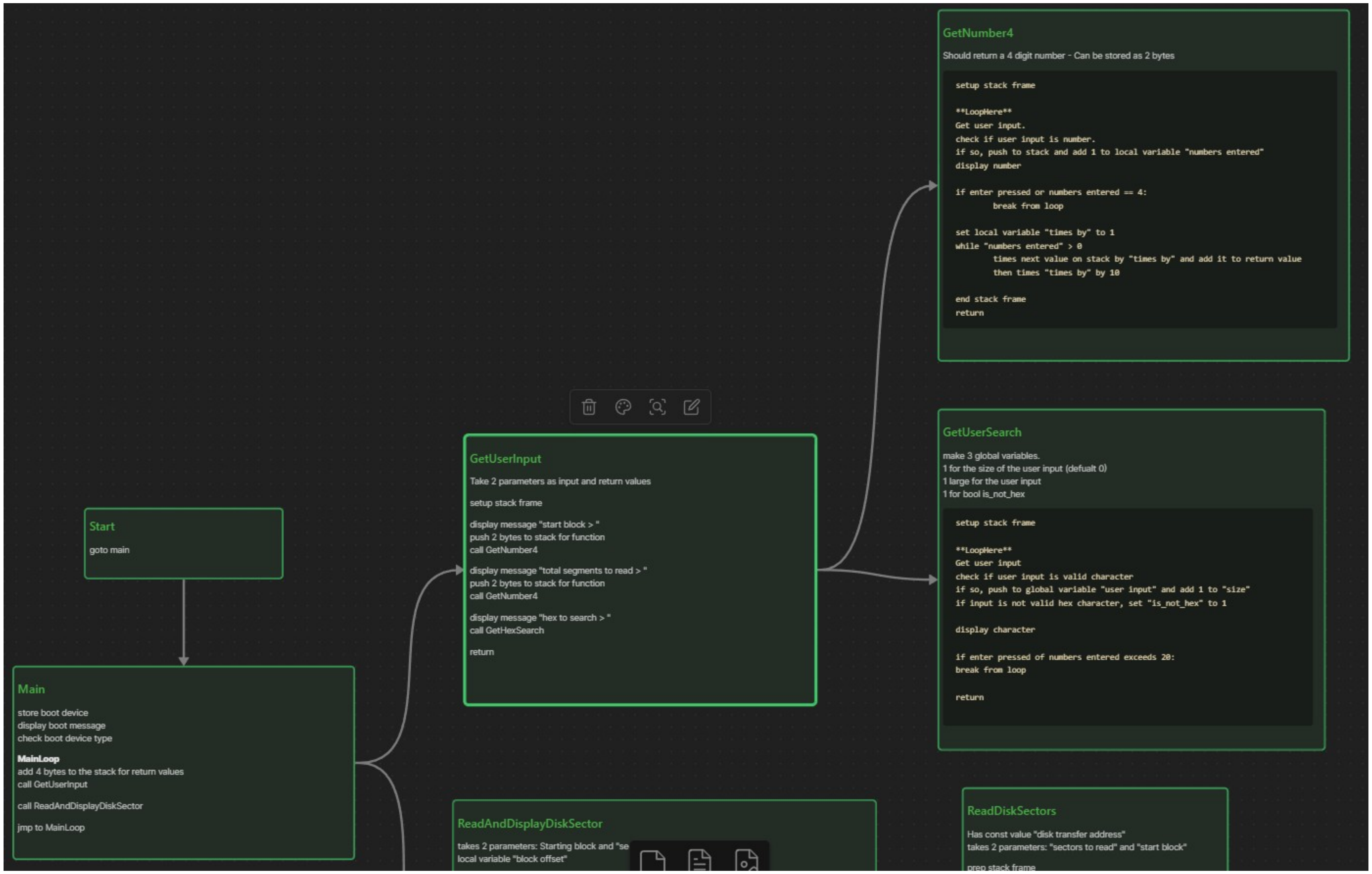
local variable: "start index", "found current index"

```
prep stack frame
push si

if "size of user input" == 0
        return

iterate through current sector
if (si) == "found current index"("user input") {
        if found_current_index == 0 then start index = si
        "found current index" += 1
}
else {
        "found current index" = 0
        si = "start_index + 1"
}

if "found current index" == "size of user input" {
        add start_index to array
}

pop si
end stack frame
return
```

DisplayHexLine

DisplayAsciiLine

## Starting section



Start

goto main

Main

store boot device
display boot message
check boot device type

**MainLoop**
add 4 bytes to the stack for return values
call GetUserInput

call ReadAndDisplayDiskSector

jmp to MainLoop

GetUserInput

Take 2 parameters as input and return values

setup stack frame

display message "start block > "
push 2 bytes to stack for function
call GetNumber4

display message "total segments to read > "
push 2 bytes to stack for function
call GetNumber4

display message "hex to search > "
call GetHexSearch

return

GetNumber4

Should return a 4 digit number - Can be stored as 2 bytes

```
setup stack frame

**LoopHere**
Get user input.
check if user input is number.
if so, push to stack and add 1 to local variable "numbers entered"
display number

if enter pressed or numbers entered == 4:
        break from loop

set local variable "times by" to 1
while "numbers entered" > 0
        times next value on stack by "times by" and add it to return value
        then times "times by" by 10

end stack frame
return
```

GetUserSearch

make 3 global variables.
1 for the size of the user input (defualt 0)
1 large for the user input
1 for bool is_not_hex

```
setup stack frame

**LoopHere**
Get user input
check if user input is valid character
if so, push to global variable "user input" and add 1 to "size"
if input is not valid hex character, set "is_not_hex" to 1

display character

if enter pressed of numbers entered exceeds 20:
break from loop

return
```

ReadAndDisplayDiskSector

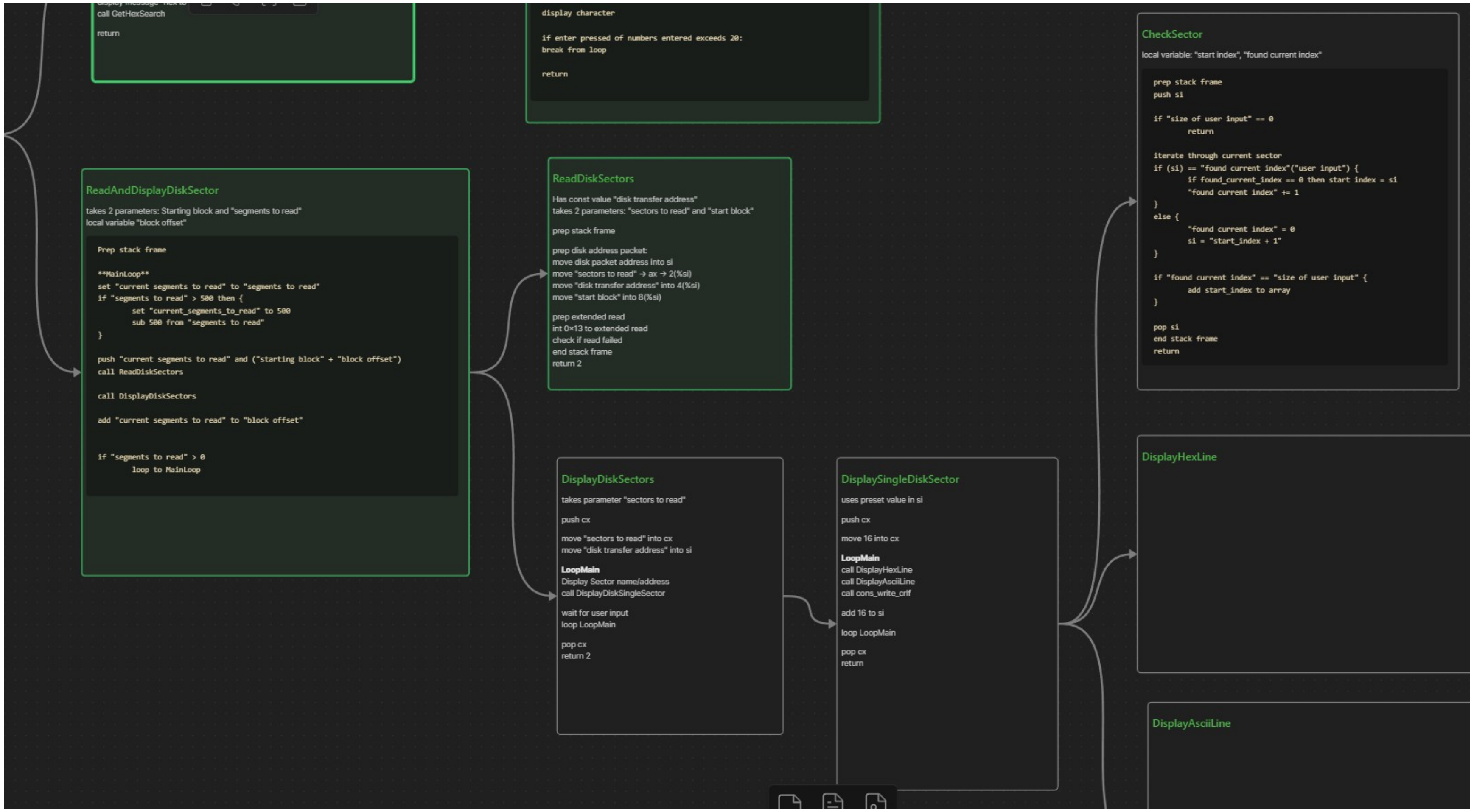
takes 2 parameters: Starting block and "se
local variable "block offset"

ReadDiskSectors

Has const value "disk transfer address"
takes 2 parameters: "sectors to read" and "start block"

prep stack frame

Displaying stuff section



call GetHexSearch

return

```
display character

if enter pressed of numbers entered exceeds 20:
break from loop

return
```

ReadAndDisplayDiskSector

takes 2 parameters: Starting block and "segments to read"
local variable "block offset"

```
Prep stack frame

**MainLoop**
set "current segments to read" to "segments to read"
if "segments to read" > 500 then {
        set "current_segments_to_read" to 500
        sub 500 from "segments to read"
}

push "current segments to read" and ("starting block" + "block offset")
call ReadDiskSectors

call DisplayDiskSectors

add "current segments to read" to "block offset"


if "segments to read" > 0
        loop to MainLoop
```

ReadDiskSectors

Has const value "disk transfer address"
takes 2 parameters: "sectors to read" and "start block"

prep stack frame

prep disk address packet:
move disk packet address into si
move "sectors to read" → ax → 2(%si)
move "disk transfer address" into 4(%si)
move "start block" into 8(%si)

prep extended read
int 0×13 to extended read
check if read failed
end stack frame
return 2

CheckSector

local variable: "start index", "found current index"

```
prep stack frame
push si

if "size of user input" == 0
        return

iterate through current sector
if (si) == "found current index"("user input") {
        if found_current_index == 0 then start index = si
        "found current index" += 1
}
else {
        "found current index" = 0
        si = "start_index + 1"
}

if "found current index" == "size of user input" {
        add start_index to array
}

pop si
end stack frame
return
```

DisplayDiskSectors

takes parameter "sectors to read"

push cx

move "sectors to read" into cx
move "disk transfer address" into si

**LoopMain**
Display Sector name/address
call DisplayDiskSingleSector

wait for user input
loop LoopMain

pop cx
return 2

DisplaySingleDiskSector

uses preset value in si

push cx

move 16 into cx

**LoopMain**
call DisplayHexLine
call DisplayAsciiLine
call cons_write_crlf

add 16 to si

loop LoopMain

pop cx
return

DisplayHexLine

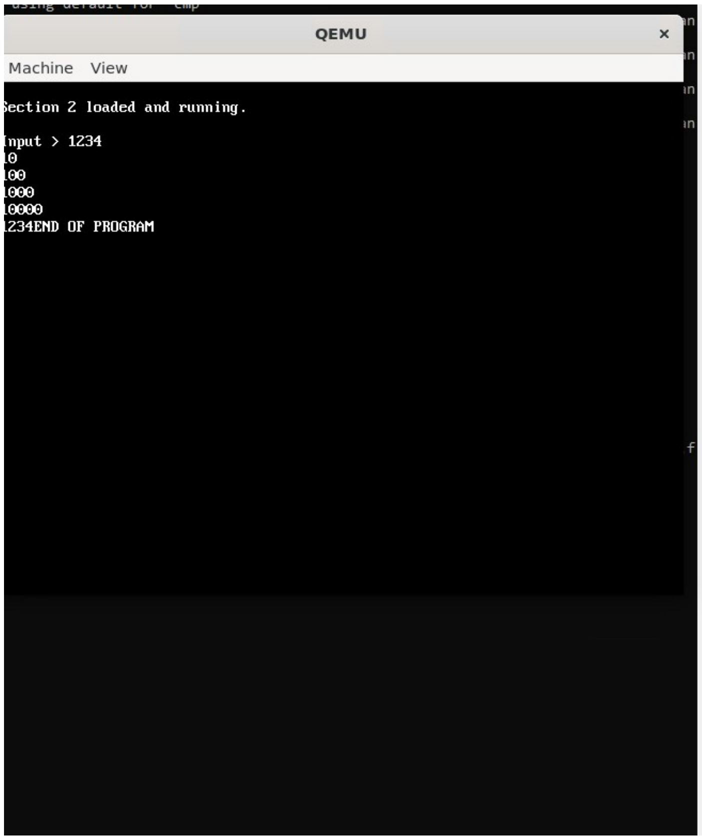
DisplayAsciiLine

15/11/23

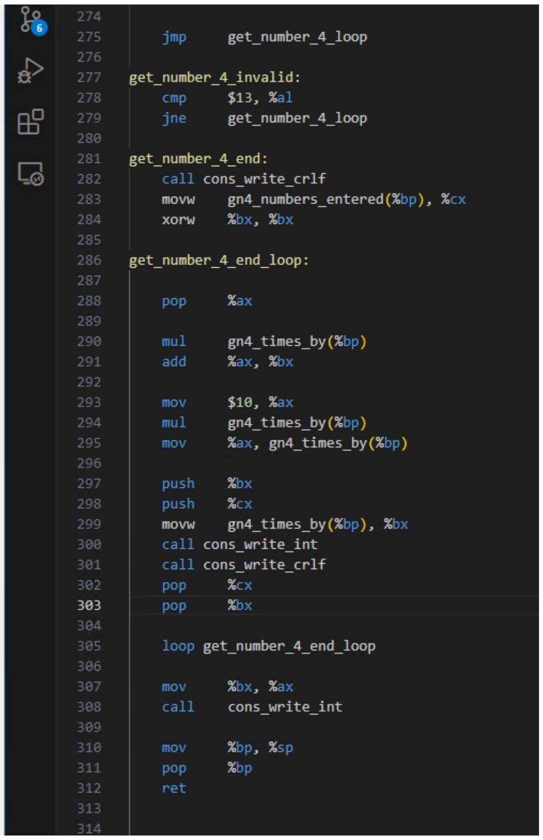
Spent most of the day on this (6-8 hours). Started the bulk implementation of the pseudocode starting with a get_number_4 function which would get a 4 digit function from the user.

I was still getting familiar with stack frames and perhaps overused them a little however with the absence of easy to understand debugging tools, It made sense to overuse them instead of underuse to try and prevent as much anomalous behaviour from ever occurring. I also started using them a lot to try and remain consistent throughout my code, adding them to all main functions however I later removed the really unneeded ones.

Creating the function was straightforwards, mainly translating the pseudocode and making sure to keep track of used registers. Initially, I got jibberish from the function when displayed which I later found was because I forgot to initialize my local variables however once fixed, I got overly large numbers which I discovered because of incorrectly set local variables. I found this by printing the multiplier with the provided cons_write_int function as followed:



```
ection 2 loaded and running.

nput > 1234
0
00
000
0000
234END OF PROGRAM
```



```
    jmp     get_number_4_loop

get_number_4_invalid:
    cmp     $13, %al
    jne     get_number_4_loop

get_number_4_end:
    call cons_write_crlf
    movw    gn4_numbers_entered(%bp), %cx
    xorw    %bx, %bx

get_number_4_end_loop:

    pop     %ax

    mul     gn4_times_by(%bp)
    add     %ax, %bx

    mov     $10, %ax
    mul     gn4_times_by(%bp)
    mov     %ax, gn4_times_by(%bp)

    push    %bx
    push    %cx
    movw    gn4_times_by(%bp), %bx
    call cons_write_int
    call cons_write_crlf
    pop     %cx
    pop     %bx

    loop get_number_4_end_loop

    mov     %bx, %ax
    call    cons_write_int

    mov     %bp, %sp
    pop     %bp
    ret
```

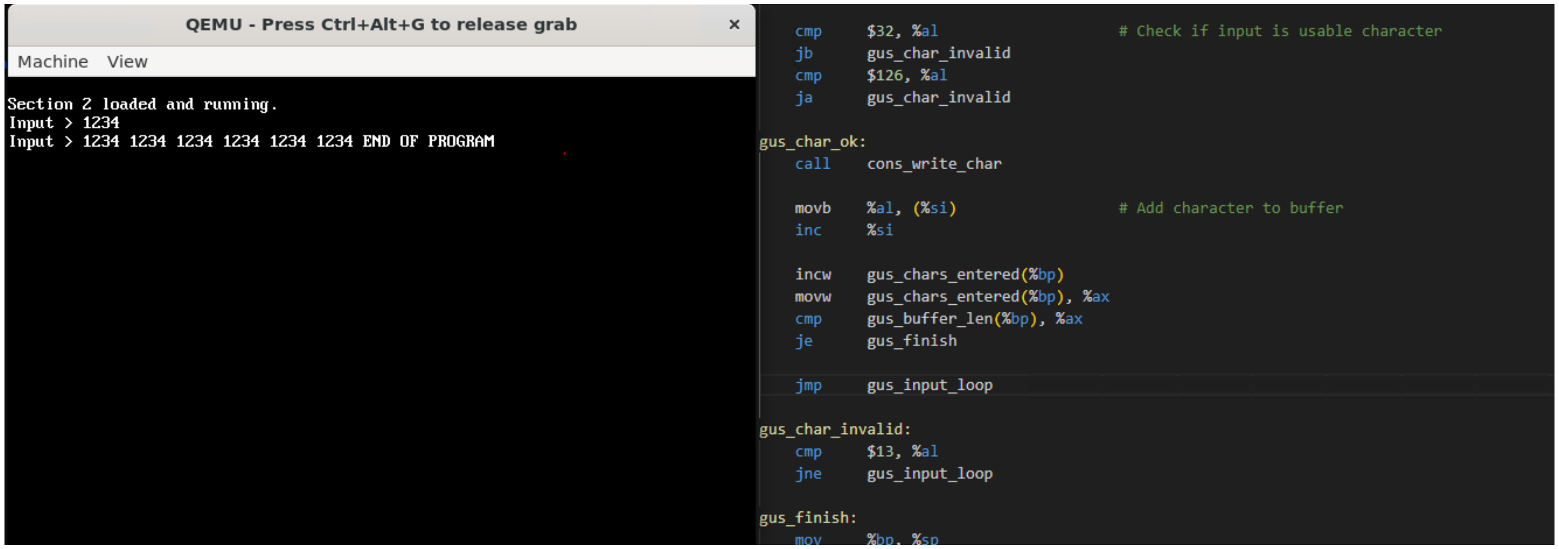
I also decided on using stack frame parameters as return values as it felt simple and easy to use/understand and looked to work well with using multiple return values in future functions.

I then implemented the get_user_search function. Similar to get_number_4, it was mainly a straight translation of pseudocode with the addition of checking the length of the UserSeachBuffer ahead of time. It would have been easier and perhaps more sensible to hardcode a value to the predetermined size of the UserSearchBuffer however I wanted to do it dynamically. My implementation should allow for any (sensible) size search buffer. I made one that was 30 characters long as that felt like it would be long enough to cover most use cases. Additionally, I removed the pseudocode regarding hexidecimal input as I might implement it later but not now.

With the addition of another function with multiple new labels and stack frame definitions, I've opted to change labels and definitions to be prefixed with the functions initials (i.e. get_user_search_finish -> gus_finish). Hopefully this will prevent any naming collisions in future and also help to identify which labels and definitions belong where in the future. It also reduces the size of some labels and definitions.

I've provided some screenshots of testing the max input size, printing the final user input and of getting the length of the user input. I often used cons_write_int in debugging to check values as well as printing the END OF PROGRAM text at different points in my code to mark when my code reached certain points.

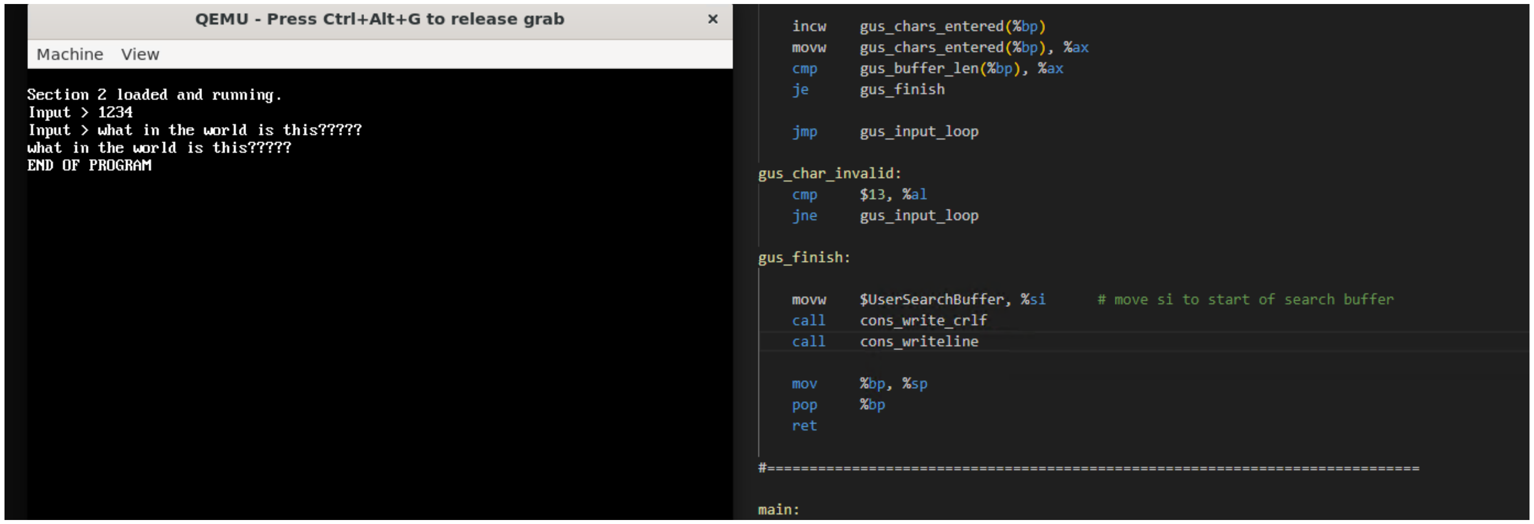
Make sure user input doesn’t exceed max size (30 in this case).



```
Section 2 loaded and running.
Input > 1234
Input > 1234 1234 1234 1234 1234 1234 END OF PROGRAM
```

```
    cmp     $32, %al                        # Check if input is usable character
    jb      gus_char_invalid
    cmp     $126, %al
    ja      gus_char_invalid

gus_char_ok:
    call    cons_write_char

    movb    %al, (%si)                      # Add character to buffer
    inc     %si

    incw    gus_chars_entered(%bp)
    movw    gus_chars_entered(%bp), %ax
    cmp     gus_buffer_len(%bp), %ax
    je      gus_finish

    jmp     gus_input_loop

gus_char_invalid:
    cmp     $13, %al
    jne     gus_input_loop

gus_finish:
    mov     %bp, %sp
```

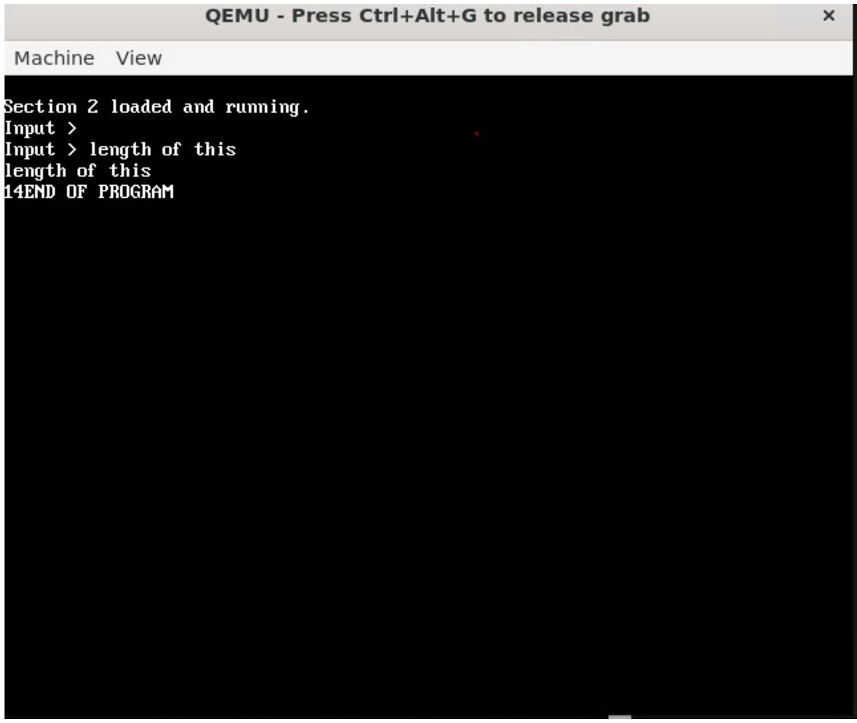
Make sure input set correctly (inputted and then re-printed)



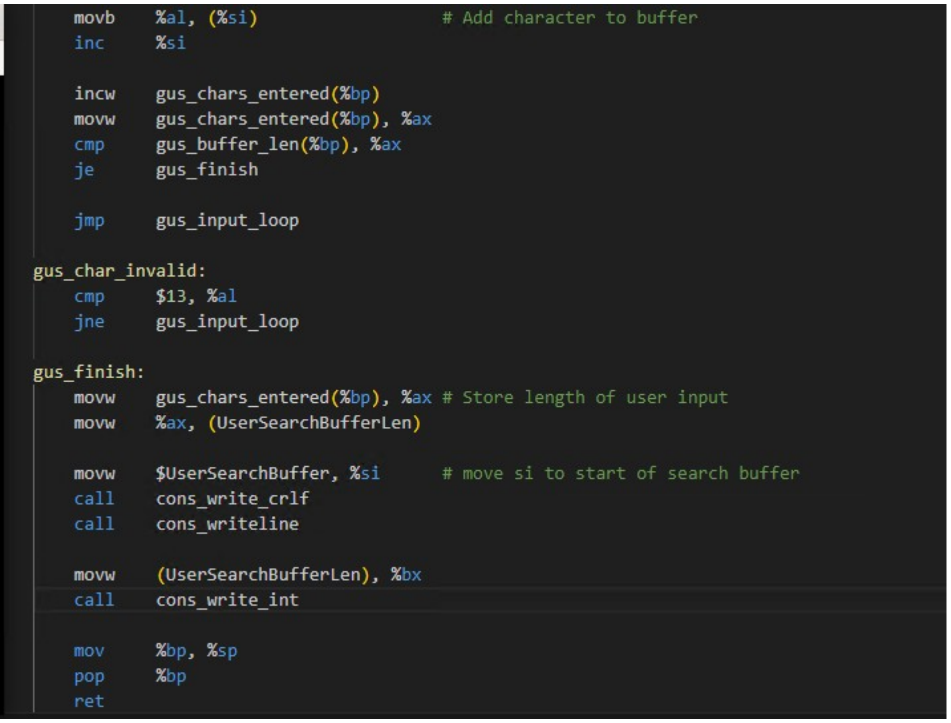
```
Section 2 loaded and running.
Input > 1234
Input > what in the world is this?????
what in the world is this?????
END OF PROGRAM
```

```
    incw    gus_chars_entered(%bp)
    movw    gus_chars_entered(%bp), %ax
    cmp     gus_buffer_len(%bp), %ax
    je      gus_finish

    jmp     gus_input_loop

gus_char_invalid:
    cmp     $13, %al
    jne     gus_input_loop

gus_finish:

    movw    $UserSearchBuffer, %si      # move si to start of search buffer
    call    cons_write_crlf
    call    cons_writeline

    mov     %bp, %sp
    pop     %bp
    ret

#===========================================================================

main:
```

Check length of user input ("length of this" -> 14 chars long)

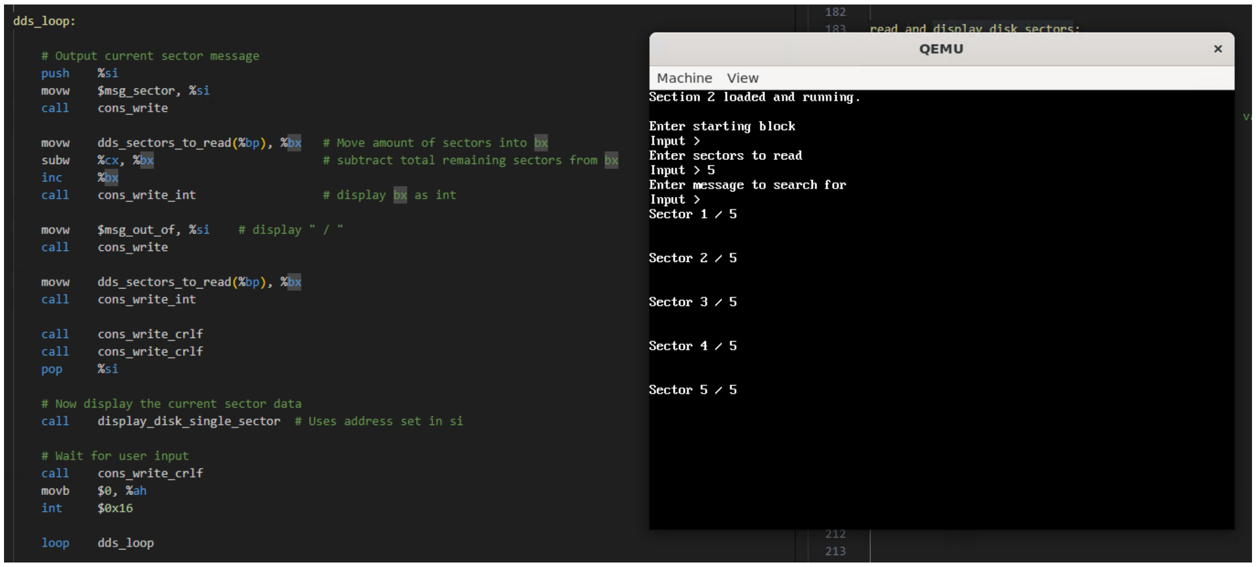


```
Section 2 loaded and running.
Input >
Input > length of this
length of this
14END OF PROGRAM
```



```
    movb    %al, (%si)                  # Add character to buffer
    inc     %si

    incw    gus_chars_entered(%bp)
    movw    gus_chars_entered(%bp), %ax
    cmp     gus_buffer_len(%bp), %ax
    je      gus_finish

    jmp     gus_input_loop

gus_char_invalid:
    cmp     $13, %al
    jne     gus_input_loop

gus_finish:
    movw    gus_chars_entered(%bp), %ax # Store length of user input
    movw    %ax, (UserSearchBufferLen)

    movw    $UserSearchBuffer, %si      # move si to start of search buffer
    call    cons_write_crlf
    call    cons_writeline

    movw    (UserSearchBufferLen), %bx
    call    cons_write_int

    mov     %bp, %sp
    pop     %bp
    ret
```

I then started on the read_and_display_disk_sector from the pseudocode. I initially skipped the read_disk_sector function and went straight to display_disk_sector function. While I don't think mentioned in the brief, I added a counter to keep track of which sector we're currently reading. This actually helped with debugging and making the output easier to follow.

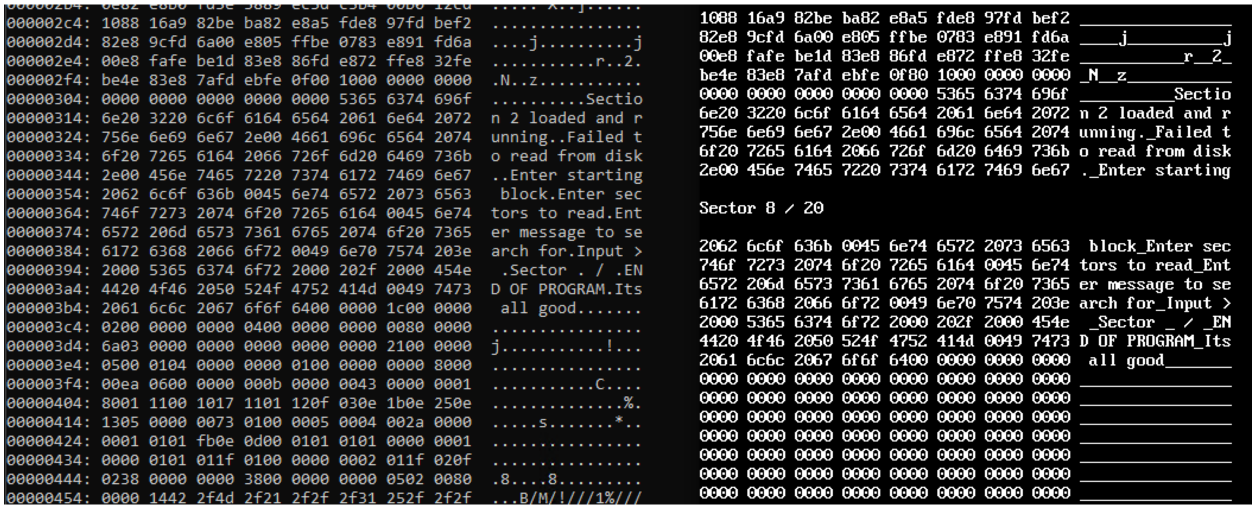


```
dds_loop:

    # Output current sector message
    push    %si
    movw    $msg_sector, %si
    call    cons_write

    movw    dds_sectors_to_read(%bp), %bx   # Move amount of sectors into bx
    subw    %cx, %bx                        # subtract total remaining sectors from bx
    inc     %bx
    call    cons_write_int                  # display bx as int

    movw    $msg_out_of, %si    # display " / "
    call    cons_write

    movw    dds_sectors_to_read(%bp), %bx
    call    cons_write_int

    call    cons_write_crlf
    call    cons_write_crlf
    pop     %si

    # Now display the current sector data
    call    display_disk_single_sector  # Uses address set in si

    # Wait for user input
    call    cons_write_crlf
    movb    $0, %ah
    int     $0x16

    loop    dds_loop
```

```
read_and_display_disk_sectors:
```

```
Section 2 loaded and running.

Enter starting block
Input >
Enter sectors to read
Input > 5
Enter message to search for
Input >
Sector 1 / 5


Sector 2 / 5


Sector 3 / 5


Sector 4 / 5


Sector 5 / 5
```

It was then straight forward to plug in the previous display disk sector function although I did rename it and modify it and add in the wait for user input functionality. Comparing against xxd output was now possible again and easier as I could read more area from memory more easily.

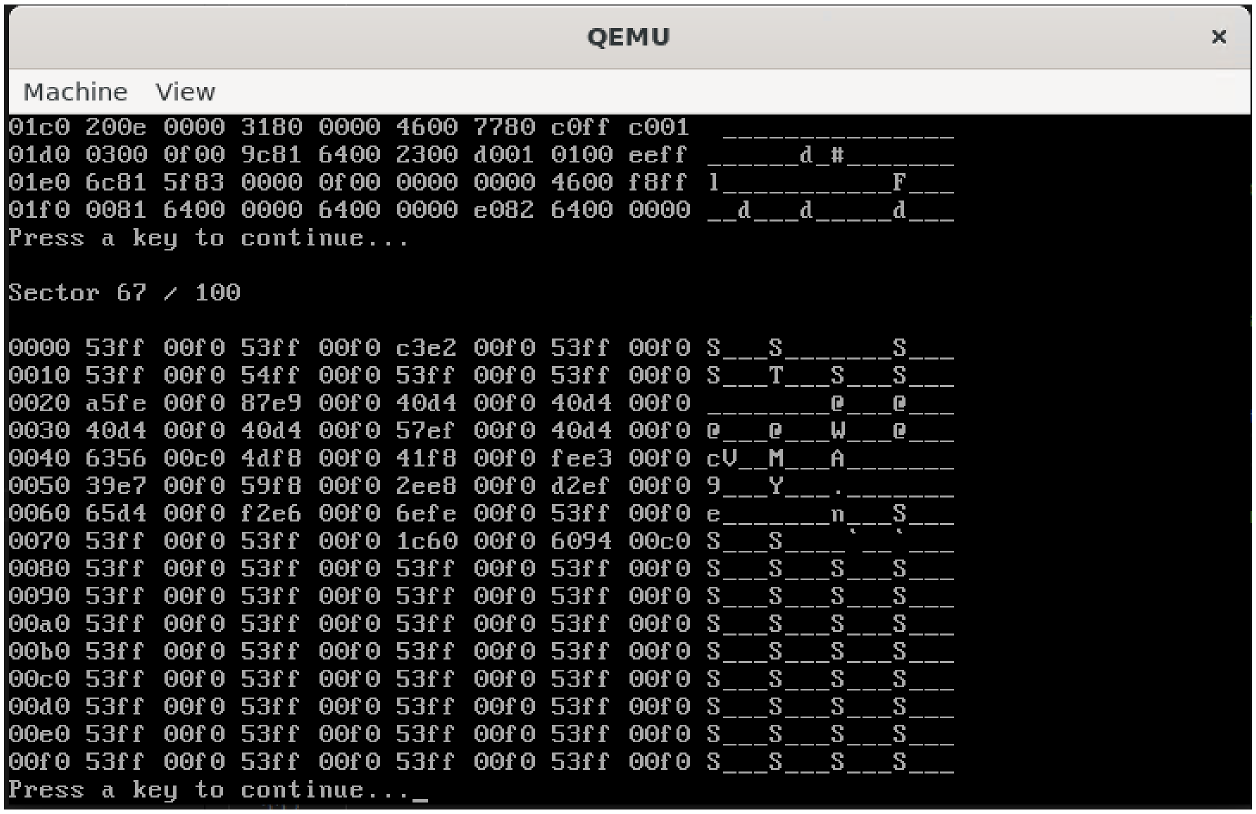


```
000002c4: 1088 16a9 82be ba82 e8a5 fde8 97fd bef2  ................
000002d4: 82e8 9cfd 6a00 e805 ffbe 0783 e891 fd6a  ....j..........j
000002e4: 00e8 fafe be1d 83e8 86fd e872 ffe8 32fe  ...........r..2.
000002f4: be4e 83e8 7afd ebfe 0f00 1000 0000 0000  .N..z...........
00000304: 0000 0000 0000 0000 0000 5365 6374 696f  ..........Sectio
00000314: 6e20 3220 6c6f 6164 6564 2061 6e64 2072  n 2 loaded and r
00000324: 756e 6e69 6e67 2e00 4661 696c 6564 2074  unning..Failed t
00000334: 6f20 7265 6164 2066 726f 6d20 6469 736b  o read from disk
00000344: 2e00 456e 7465 7220 7374 6172 7469 6e67  ..Enter starting
00000354: 2062 6c6f 636b 0045 6e74 6572 2073 6563   block.Enter sec
00000364: 746f 7273 2074 6f20 7265 6164 0045 6e74  tors to read.Ent
00000374: 6572 206d 6573 7361 6765 2074 6f20 7365  er message to se
00000384: 6172 6368 2066 6f72 0049 6e70 7574 203e  arch for.Input >
00000394: 2000 5365 6374 6f72 2000 202f 2000 454e   .Sector . / .EN
000003a4: 4420 4f46 2050 524f 4752 414d 0049 7473  D OF PROGRAM.Its
000003b4: 2061 6c6c 2067 6f6f 6400 0000 1c00 0000   all good.......
000003c4: 0200 0000 0000 0400 0000 0000 0080 0000  ................
000003d4: 6a03 0000 0000 0000 0000 0000 2100 0000  j...........!...
000003e4: 0500 0104 0000 0000 0100 0000 0000 8000  ................
000003f4: 00ea 0600 0000 000b 0000 0043 0000 0001  ...........C....
00000404: 8001 1100 1017 1101 120f 030e 1b0e 250e  ..............%.
00000414: 1305 0000 0073 0100 0005 0004 002a 0000  .....s.......*..
00000424: 0001 0101 fb0e 0d00 0101 0101 0000 0001  ................
00000434: 0000 0101 011f 0100 0000 0002 011f 020f  ................
00000444: 0238 0000 0000 3800 0000 0000 0502 0080  .8....8.........
00000454: 0000 1442 2f4d 2f21 2f2f 2f31 252f 2f2f  ...B/M/!///1%///
```

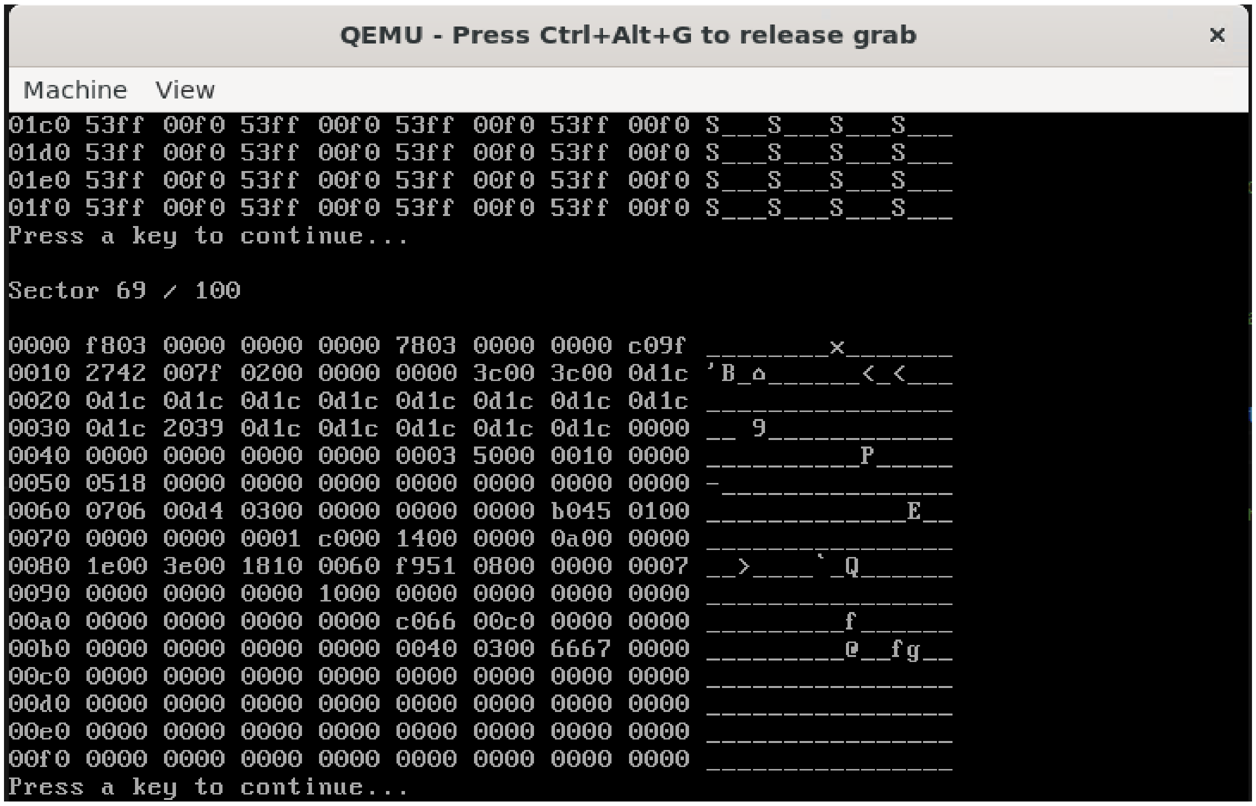
```
1088 16a9 82be ba82 e8a5 fde8 97fd bef2 ________________
82e8 9cfd 6a00 e805 ffbe 0783 e891 fd6a ____j__________j
00e8 fafe be1d 83e8 86fd e872 ffe8 32fe ___________r__2_
be4e 83e8 7afd ebfe 0f80 1000 0000 0000 _N__z___________
0000 0000 0000 0000 0000 5365 6374 696f __________Sectio
6e20 3220 6c6f 6164 6564 2061 6e64 2072 n 2 loaded and r
756e 6e69 6e67 2e00 4661 696c 6564 2074 unning._Failed t
6f20 7265 6164 2066 726f 6d20 6469 736b o read from disk
2e00 456e 7465 7220 7374 6172 7469 6e67 ._Enter starting

Sector 8 / 20

2062 6c6f 636b 0045 6e74 6572 2073 6563  block_Enter sec
746f 7273 2074 6f20 7265 6164 0045 6e74 tors to read_Ent
6572 206d 6573 7361 6765 2074 6f20 7365 er message to se
6172 6368 2066 6f72 0049 6e70 7574 203e arch for_Input >
2000 5365 6374 6f72 2000 202f 2000 454e  _Sector _ / _EN
4420 4f46 2050 524f 4752 414d 0049 7473 D OF PROGRAM_Its
2061 6c6c 2067 6f6f 6400 0000 0000 0000  all good_______
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
0000 0000 0000 0000 0000 0000 0000 0000 ________________
```

I added the sector offset after this using the cons_write_hex and found that the bytes were flipped around. To fix this, I moved the code to flip the endianness from cons_write_hex to cons_write_hex_line.

Using this functionality, I played around with reading from memory and found some memory in use in sector 66-69 (sector 0 being 0x7c00). After doing some quick maths, I think 66 sectors after 0x7c00 is around 0x10000 so I assume this is the stack. This means that when loading from the disk, I can load roughly around 50 sectors of data at once after bootblock2.



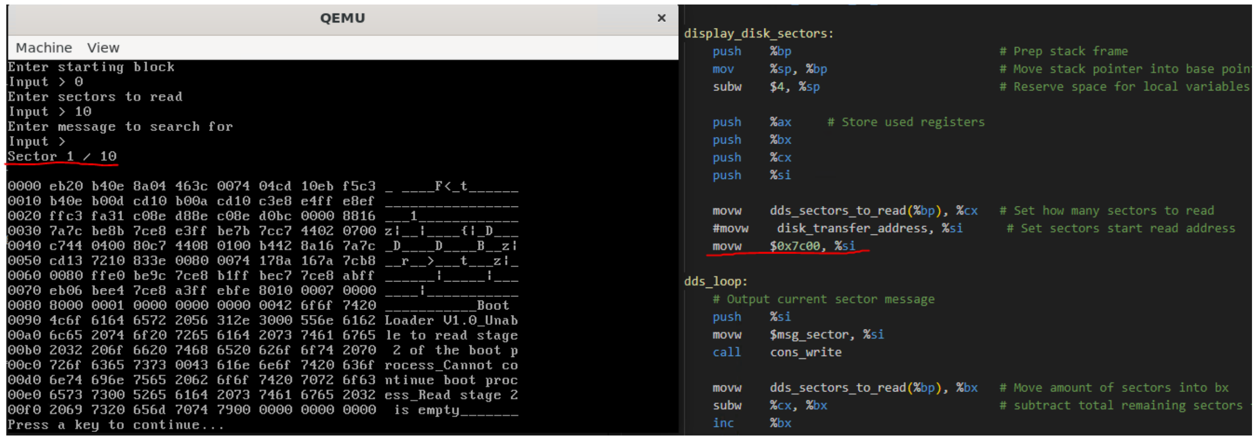
```
01c0 200e 0000 3180 0000 4600 7780 c0ff c001  _______________
01d0 0300 0f00 9c81 6400 2300 d001 0100 eeff ______d_#_______
01e0 6c81 5f83 0000 0f00 0000 0000 4600 f8ff l___________F___
01f0 0081 6400 0000 6400 0000 e082 6400 0000 __d___d_____d___
Press a key to continue...

Sector 67 / 100

0000 53ff 00f0 53ff 00f0 c3e2 00f0 53ff 00f0 S___S_______S___
0010 53ff 00f0 54ff 00f0 53ff 00f0 53ff 00f0 S___T___S___S___
0020 a5fe 00f0 87e9 00f0 40d4 00f0 40d4 00f0 ________@___@___
0030 40d4 00f0 40d4 00f0 57ef 00f0 40d4 00f0 @___@___W___@___
0040 6356 00c0 4df8 00f0 41f8 00f0 fee3 00f0 cV__M___A_______
0050 39e7 00f0 59f8 00f0 2ee8 00f0 d2ef 00f0 9___Y___._______
0060 65d4 00f0 f2e6 00f0 6efe 00f0 53ff 00f0 e_______n___S___
0070 53ff 00f0 53ff 00f0 1c60 00f0 6094 00c0 S___S____`__`___
0080 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
0090 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00a0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00b0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00c0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00d0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00e0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
00f0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
Press a key to continue..._
```



```
01c0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
01d0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
01e0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
01f0 53ff 00f0 53ff 00f0 53ff 00f0 53ff 00f0 S___S___S___S___
Press a key to continue...

Sector 69 / 100

0000 f803 0000 0000 0000 7803 0000 0000 c09f ________x_______
0010 2742 007f 0200 0000 0000 3c00 3c00 0d1c 'B_⌂______<_<___
0020 0d1c 0d1c 0d1c 0d1c 0d1c 0d1c 0d1c 0d1c ________________
0030 0d1c 2039 0d1c 0d1c 0d1c 0d1c 0d1c 0000 __ 9____________
0040 0000 0000 0000 0000 0003 5000 0010 0000 __________P_____
0050 0518 0000 0000 0000 0000 0000 0000 0000 -_______________
0060 0706 00d4 0300 0000 0000 0000 b045 0100 _____________E__
0070 0000 0000 0001 c000 1400 0000 0a00 0000 ________________
0080 1e00 3e00 1810 0060 f951 0800 0000 0007 __>____`_Q______
0090 0000 0000 0000 1000 0000 0000 0000 0000 ________________
00a0 0000 0000 0000 0000 c066 00c0 0000 0000 _________f______
00b0 0000 0000 0000 0000 0040 0300 6667 0000 _________@__fg__
00c0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
00d0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
00e0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
00f0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
Press a key to continue...
```

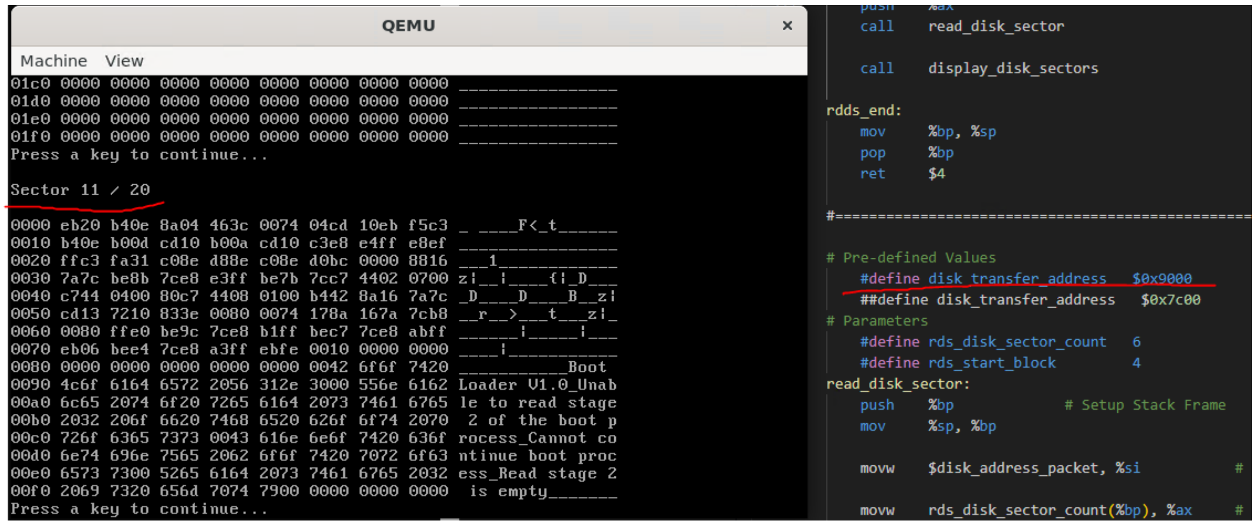
In the read_and_display_disk_sector, I opted to read 30 sectors at a time to be safe. I initially used an address like 0xA000 to load from disk into however whenever I did, I would then scan through memory as done previously and find where it was loaded, bringing it closer to 0x8000 when I saw I could. I eventually settled on 0x9000.

The following screenshots show the starting sectors of bootasm1 and then scolling down to bootasm1 loaded into address 0x9000. This method tells me that the data was loaded successfully and into the right place.



```
Enter starting block
Input > 0
Enter sectors to read
Input > 10
Enter message to search for
Input >
Sector 1 / 10

0000 eb20 b40e 8a04 463c 0074 04cd 10eb f5c3 _ ____F<_t______
0010 b40e b00d cd10 b00a cd10 c3e8 e4ff e8ef ________________
0020 ffc3 fa31 c08e d88e c08e d0bc 0000 8816 ___1____________
0030 7a7c be8b 7ce8 e3ff be7b 7cc7 4402 0700 z¦__¦____{¦_D___
0040 c744 0400 80c7 4408 0100 b442 8a16 7a7c _D____D____B__z¦
0050 cd13 7210 833e 0080 0074 178a 167a 7cb8 __r__>___t___z¦_
0060 0080 ffe0 be9c 7ce8 b1ff bec7 7ce8 abff _______¦_____¦___
0070 eb06 bee4 7ce8 a3ff ebfe 8010 0007 0000 ____¦___________
0080 8000 0001 0000 0000 0000 0042 6f6f 7420 ____________Boot
0090 4c6f 6164 6572 2056 312e 3000 556e 6162 Loader V1.0_Unab
00a0 6c65 2074 6f20 7265 6164 2073 7461 6765 le to read stage
00b0 2032 206f 6620 7468 6520 626f 6f74 2070  2 of the boot p
00c0 726f 6365 7373 0043 616e 6e6f 7420 636f rocess_Cannot co
00d0 6e74 696e 7565 2062 6f6f 7420 7072 6f63 ntinue boot proc
00e0 6573 7300 5265 6164 2073 7461 6765 2032 ess_Read stage 2
00f0 2069 7320 656d 7074 7900 0000 0000 0000  is empty_______
Press a key to continue...
```

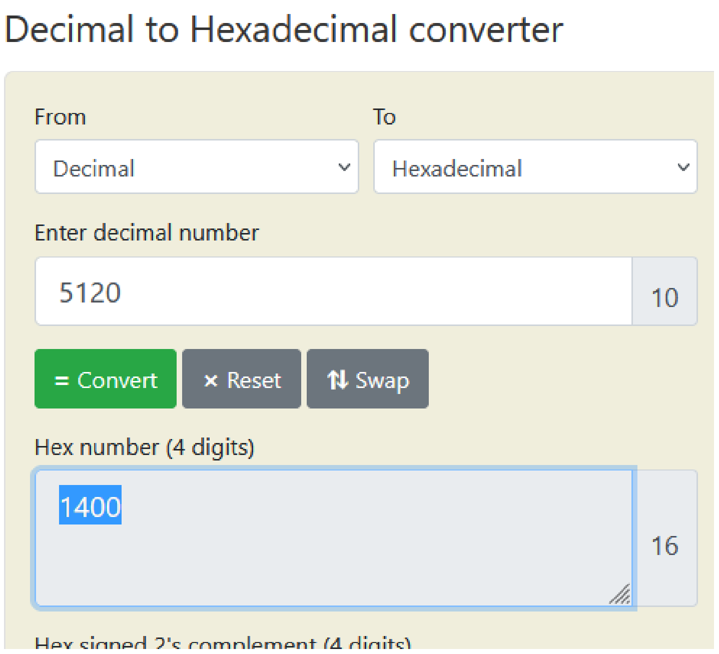
```
display_disk_sectors:
    push    %bp                                 # Prep stack frame
    mov     %sp, %bp                            # Move stack pointer into base poin
    subw    $4, %sp                             # Reserve space for local variables

    push    %ax     # Store used registers
    push    %bx
    push    %cx
    push    %si

    movw    dds_sectors_to_read(%bp), %cx   # Set how many sectors to read
    #movw    disk_transfer_address, %si      # Set sectors start read address
    movw    $0x7c00, %si

dds_loop:
    # Output current sector message
    push    %si
    movw    $msg_sector, %si
    call    cons_write

    movw    dds_sectors_to_read(%bp), %bx   # Move amount of sectors into bx
    subw    %cx, %bx                        # subtract total remaining sectors
    inc     %bx
```



```
01c0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
01d0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
01e0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
01f0 0000 0000 0000 0000 0000 0000 0000 0000 ________________
Press a key to continue...

Sector 11 / 20

0000 eb20 b40e 8a04 463c 0074 04cd 10eb f5c3 _ ____F<_t______
0010 b40e b00d cd10 b00a cd10 c3e8 e4ff e8ef ________________
0020 ffc3 fa31 c08e d88e c08e d0bc 0000 8816 ___1____________
0030 7a7c be8b 7ce8 e3ff be7b 7cc7 4402 0700 z¦__¦____{¦_D___
0040 c744 0400 80c7 4408 0100 b442 8a16 7a7c _D____D____B__z¦
0050 cd13 7210 833e 0080 0074 178a 167a 7cb8 __r__>___t___z¦_
0060 0080 ffe0 be9c 7ce8 b1ff bec7 7ce8 abff _______¦_____¦___
0070 eb06 bee4 7ce8 a3ff ebfe 0010 0000 0000 ____¦___________
0080 0000 0000 0000 0000 0000 0042 6f6f 7420 ____________Boot
0090 4c6f 6164 6572 2056 312e 3000 556e 6162 Loader V1.0_Unab
00a0 6c65 2074 6f20 7265 6164 2073 7461 6765 le to read stage
00b0 2032 206f 6620 7468 6520 626f 6f74 2070  2 of the boot p
00c0 726f 6365 7373 0043 616e 6e6f 7420 636f rocess_Cannot co
00d0 6e74 696e 7565 2062 6f6f 7420 7072 6f63 ntinue boot proc
00e0 6573 7300 5265 6164 2073 7461 6765 2032 ess_Read stage 2
00f0 2069 7320 656d 7074 7900 0000 0000 0000  is empty_______
Press a key to continue...
```

```
    push    %ax
    call    read_disk_sector

    call    display_disk_sectors

rdds_end:
    mov     %bp, %sp
    pop     %bp
    ret     $4

#==============================================

# Pre-defined Values
    #define disk_transfer_address    $0x9000
    ##define disk_transfer_address    $0x7c00
# Parameters
    #define rds_disk_sector_count    6
    #define rds_start_block          4
read_disk_sector:
    push    %bp                 # Setup Stack Frame
    mov     %sp, %bp

    movw    $disk_address_packet, %si               #

    movw    rds_disk_sector_count(%bp), %ax      #
```

Sector 11 (10 sectors later), after some quick maths as followed, does appear to be the correct sector which further confirms that the sector counter is working as intended.



Decimal to Hexadecimal converter

From: Decimal

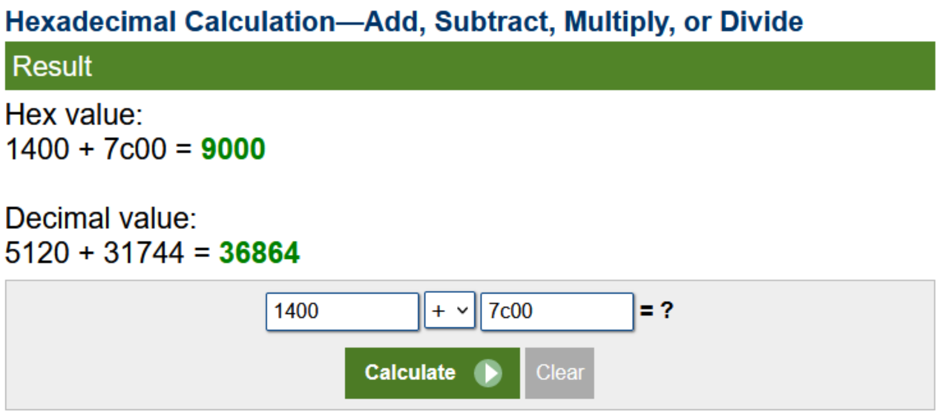
To: Hexadecimal

Enter decimal number

5120 10

Hex number (4 digits)

1400 16



Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:
1400 + 7c00 = **9000**

Decimal value:
5120 + 31744 = **36864**

1400 + 7c00 = ?

I had to change some of the code when displaying which sector we're currently in to allow for the wrap around 30 sector limit implemented in read_and_display_disk_sectors function as if you were reading 40 sectors, it would always read 1/30 and then when it reached 30/30 it would go to 1/10 for the last few. I created some more variables and changed the maths to make this work.

21/11/23

Spent the same amount of time, around 6-8 hours finishing this off.

I was looking to implement the search functionality today. I was looking into how to display different colors in the terminal and saw that the previously used int 10h, ah 0Eh teletype output had a foreground color parameter but was only enabled in graphics mode. Looking into it, I found a table here

https://www.minuszerodegrees.net/video/bios_video_modes.htm

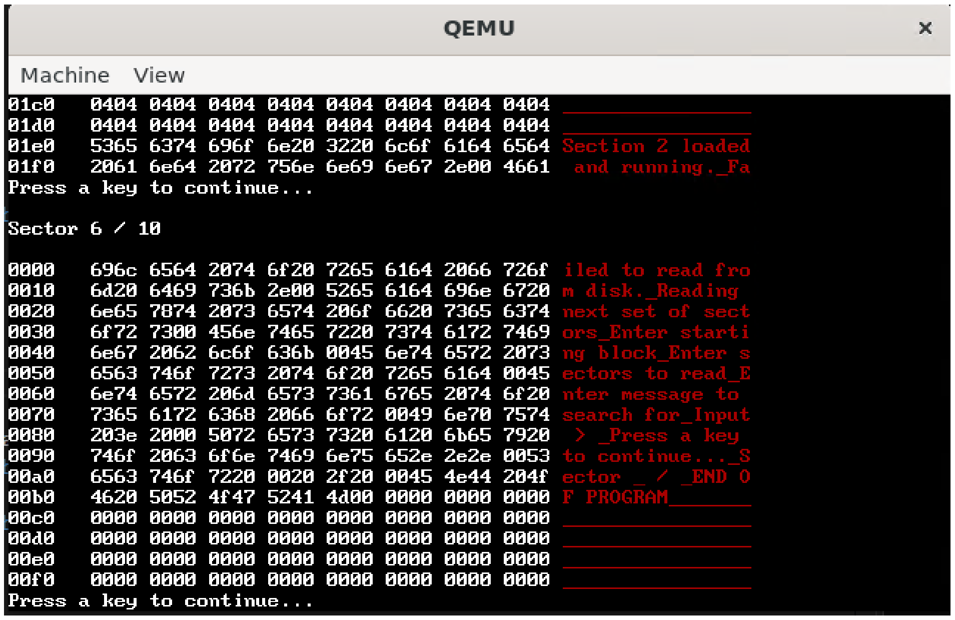
which had a list of different text/graphics modes that could be used with int 10h, ah 0h. I went through some of them and found some with strange resolutions however settled on either 0x10h or 0x12h. Both seemed to delay the output functionality when scrolling sectors however 0x10h felt like it was quicker and so I went with that.

I had to change the input register for cons_write_hex to dx as it used bx previously which messed with the colors displayed as bl is the input parameter for the teletype output color.

I initially implemented the check_sector function which I renamed scan_single_sector roughly as I had in pseudocode however decided for the array structure to use a sector worth of data and whenever I found a match in a sector, I would mark the index into the array that was equal to the index into sector. Then, in the cons_write_ascii_line, I would check the current character index against the array index, and use the value stored in there as a value for the color.

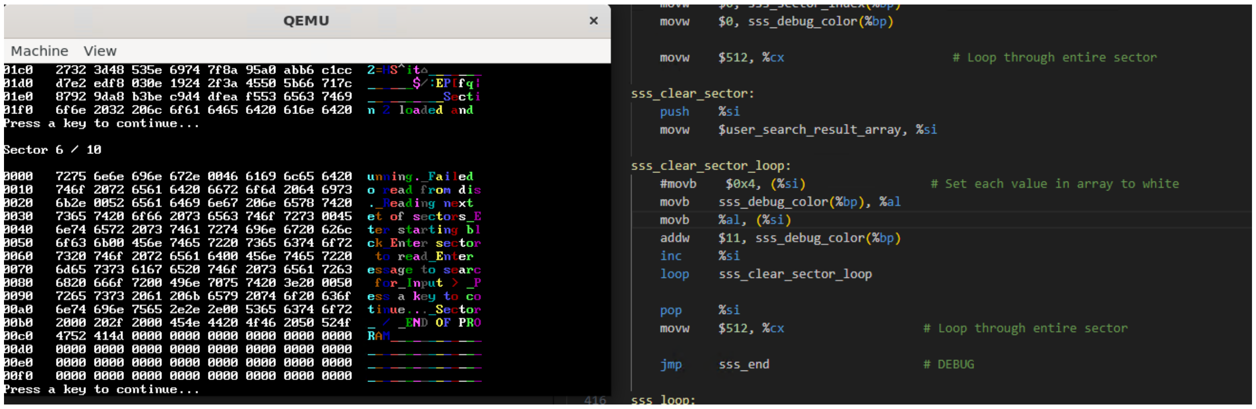
This approach feels somewhat inefficient, requiring a good sized chunk of memory and reading from it ever character printed however it was the first and easiest thing that came to mind. This approach also only works for scanning a single sector and not across them however that is what's asked in the brief so I felt like that was fine.

Setting the array to a single color gave me this result:



```
01c0   0404 0404 0404 0404 0404 0404 0404 0404 ________________
01d0   0404 0404 0404 0404 0404 0404 0404 0404 ________________
01e0   5365 6374 696f 6e20 3220 6c6f 6164 6564 Section 2 loaded
01f0   2061 6e64 2072 756e 6e69 6e67 2e00 4661  and running._Fa
Press a key to continue...

Sector 6 / 10

0000   696c 6564 2074 6f20 7265 6164 2066 726f iled to read fro
0010   6d20 6469 736b 2e00 5265 6164 696e 6720 m disk._Reading
0020   6e65 7874 2073 6574 206f 6620 7365 6374 next set of sect
0030   6f72 7300 456e 7465 7220 7374 6172 7469 ors_Enter starti
0040   6e67 2062 6c6f 636b 0045 6e74 6572 2073 ng block_Enter s
0050   6563 746f 7273 2074 6f20 7265 6164 0045 ectors to read_E
0060   6e74 6572 206d 6573 7361 6765 2074 6f20 nter message to
0070   7365 6172 6368 2066 6f72 0049 6e70 7574 search for_Input
0080   203e 2000 5072 6573 7320 6120 6b65 7920  > _Press a key
0090   746f 2063 6f6e 7469 6e75 652e 2e2e 0053 to continue..._S
00a0   6563 746f 7220 0020 2f20 0045 4e44 204f ector _ / _END O
00b0   4620 5052 4f47 5241 4d00 0000 0000 0000 F PROGRAM_______
00c0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00d0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00e0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00f0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
Press a key to continue...
```

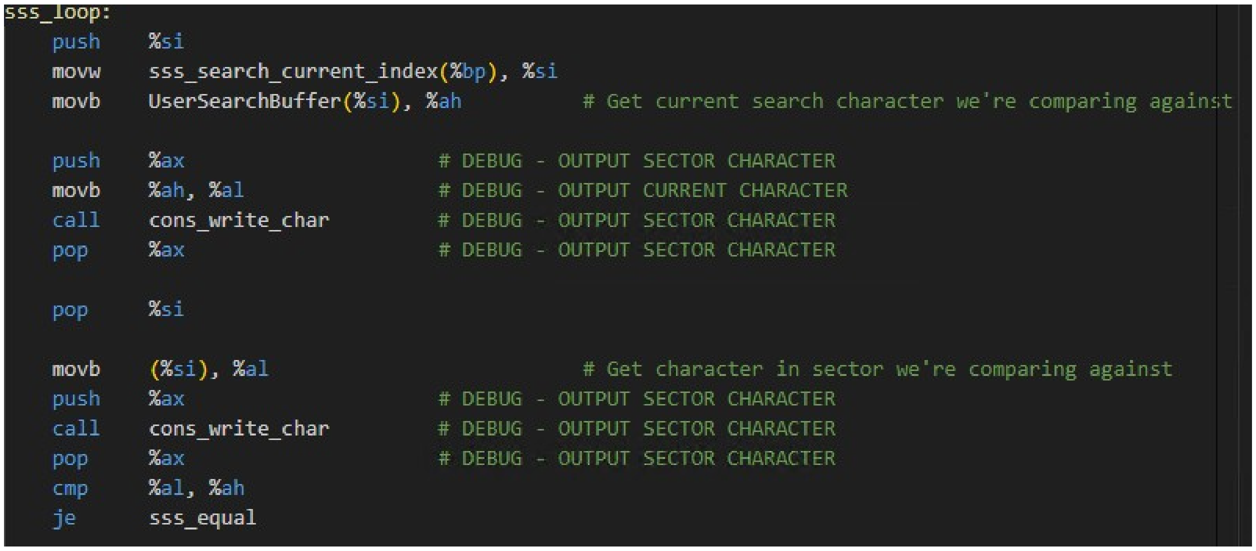
By iterating over the array and offsetting each value by 1 (11 in this screenshot), I could show that the cons_write_ascii_line was reading the color from the array correctly.



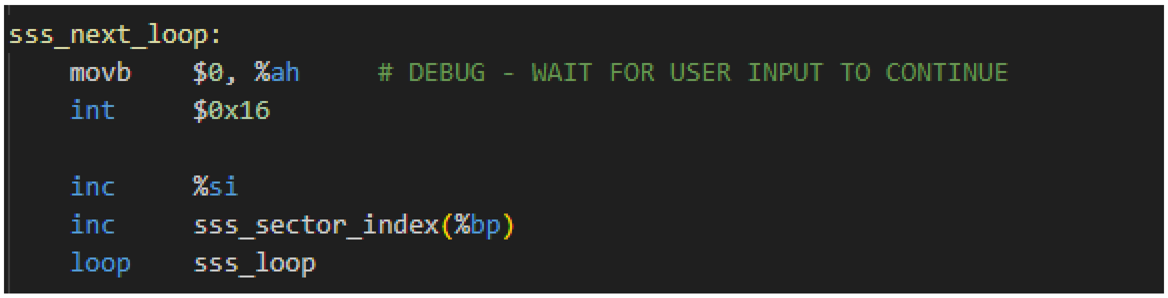
```
01c0   2732 3d48 535e 6974 7f8a 95a0 abb6 c1cc  2=HS^it⌂________
01d0   d7e2 edf8 030e 1924 2f3a 4550 5b66 717c  ______$/:EP[fq|
01e0   8792 9da8 b3be c9d4 dfea f553 6563 7469  __________Secti
01f0   6f6e 2032 206c 6f61 6465 6420 616e 6420  n 2 loaded and
Press a key to continue...

Sector 6 / 10

0000   7275 6e6e 696e 672e 0046 6169 6c65 6420  unning._Failed
0010   746f 2072 6561 6420 6672 6f6d 2064 6973  o read from dis
0020   6b2e 0052 6561 6469 6e67 206e 6578 7420  ._Reading next
0030   7365 7420 6f66 2073 6563 746f 7273 0045  et of sectors_E
0040   6e74 6572 2073 7461 7274 696e 6720 626c  ter starting bl
0050   6f63 6b00 456e 7465 7220 7365 6374 6f72  ck_Enter sector
0060   7320 746f 2072 6561 6400 456e 7465 7220   to read_Enter
0070   6d65 7373 6167 6520 746f 2073 6561 7263  essage to searc
0080   6820 666f 7200 496e 7075 7420 3e20 0050   for_Input > _P
0090   7265 7373 2061 206b 6579 2074 6f20 636f  ess a key to co
00a0   6e74 696e 7565 2e2e 2e00 5365 6374 6f72  tinue..._Sector
00b0   2000 202f 2000 454e 4420 4f46 2050 524f  _ / _END OF PRO
00c0   4752 414d 0000 0000 0000 0000 0000 0000  RAM____________
00d0   0000 0000 0000 0000 0000 0000 0000 0000  ________________
00e0   0000 0000 0000 0000 0000 0000 0000 0000  ________________
00f0   0000 0000 0000 0000 0000 0000 0000 0000  ________________
Press a key to continue...
```

```
    movw    $0, sss_debug_color(%bp)

    movw    $512, %cx                       # Loop through entire sector

sss_clear_sector:
    push    %si
    movw    $user_search_result_array, %si

sss_clear_sector_loop:
    #movb    $0x4, (%si)                  # Set each value in array to white
    movb    sss_debug_color(%bp), %al
    movb    %al, (%si)
    addw    $11, sss_debug_color(%bp)
    inc     %si
    loop    sss_clear_sector_loop

    pop     %si
    movw    $512, %cx                    # Loop through entire sector

    jmp     sss_end                      # DEBUG

sss_loop:
```

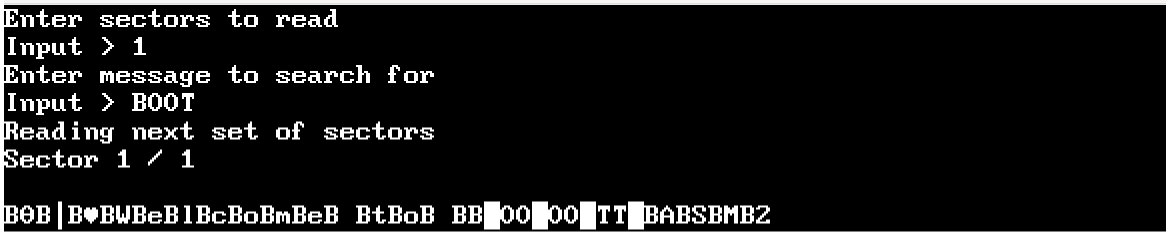
Previously, I had been setting the array and then jumping to the end of the scan_single_sector function for debugging purposes. When removing the jump, the code would not respond. To debug this, I began printing the character we're for, then the character we're checking against, then both, then waiting for user input between checks, displaing a block char whenever a match was found then finally reducing how far through the sector we scanned from 512 to 22.



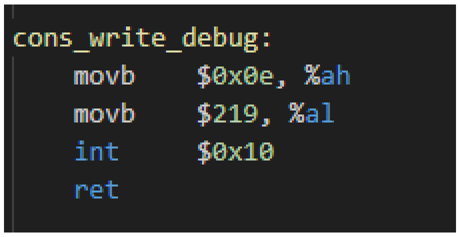
```
sss_loop:
    push    %si
    movw    sss_search_current_index(%bp), %si
    movb    UserSearchBuffer(%si), %ah          # Get current search character we're comparing against

    push    %ax                     # DEBUG - OUTPUT SECTOR CHARACTER
    movb    %ah, %al                # DEBUG - OUTPUT CURRENT CHARACTER
    call    cons_write_char         # DEBUG - OUTPUT SECTOR CHARACTER
    pop     %ax                     # DEBUG - OUTPUT SECTOR CHARACTER

    pop     %si

    movb    (%si), %al                          # Get character in sector we're comparing against
    push    %ax                     # DEBUG - OUTPUT SECTOR CHARACTER
    call    cons_write_char         # DEBUG - OUTPUT SECTOR CHARACTER
    pop     %ax                     # DEBUG - OUTPUT SECTOR CHARACTER
    cmp     %al, %ah
    je      sss_equal
```



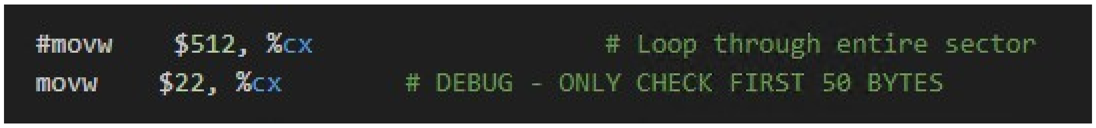
```
sss_next_loop:
    movb    $0, %ah      # DEBUG - WAIT FOR USER INPUT TO CONTINUE
    int     $0x16

    inc     %si
    inc     sss_sector_index(%bp)
    loop    sss_loop
```



```
Enter sectors to read
Input > 1
Enter message to search for
Input > BOOT
Reading next set of sectors
Sector 1 / 1

BΘB|B♥BWBeBlBcBoBmBeB BtBoB BB█OO█OO█TT█BABSBMB2
```

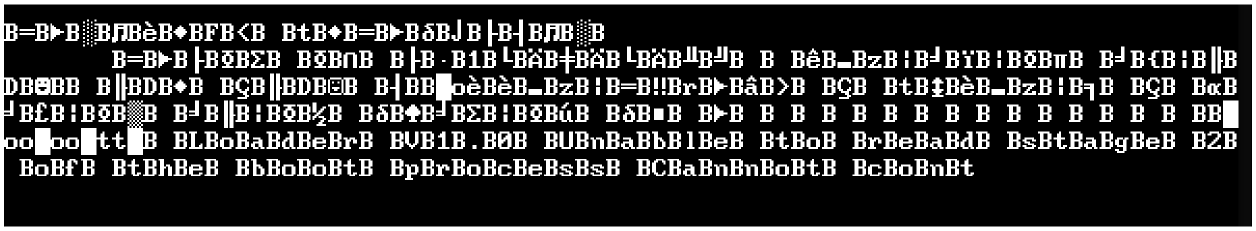


```
cons_write_debug:
    movb    $0x0e, %ah
    movb    $219, %al
    int     $0x10
    ret
```

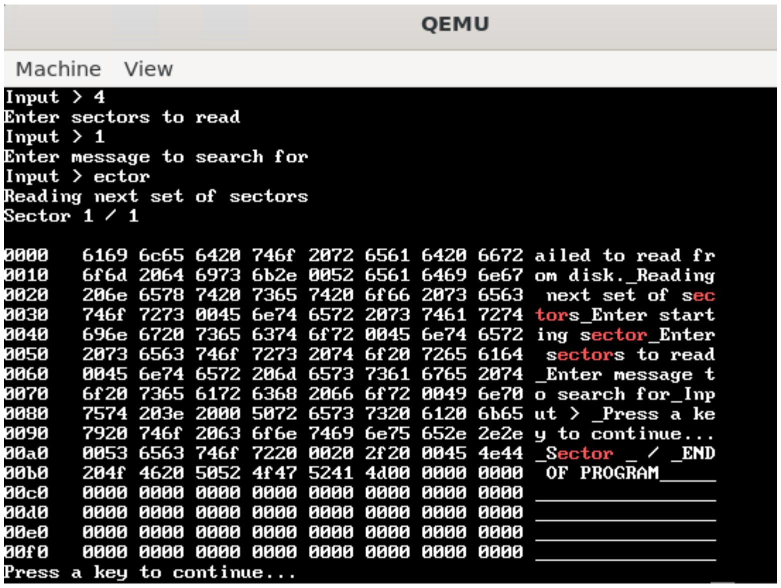


```
  #movw    $512, %cx                  # Loop through entire sector
  movw    $22, %cx           # DEBUG - ONLY CHECK FIRST 50 BYTES
```

We were printing the ascii characters below 32 however this was fine for debugging however make the results look a little strange. I eventually found a problem with the maths involved as well as realizing that the cons_write_char was overwriting ax which I was using to compare the values. After fixing all of that, debugging the first few lines worked as intended. The solid white blocks below are the matches. We can see that when looking for the term "Boot", it was found successfully on the 4th/5th row.



```
B=B▶B░BЛBèB♦BFB<B BtB♦B=B▶BδBJB├B┤BЛB░B
       B=B▶B├BΣBΣB BΣB∩B B├B·B1B└BÄB╪BÄB└BÄB╨B╝B B BêB▬BzB¦B┘BïB¦BΣBπB B┘B{B¦B║B
DB☻BB B║BDB♦B BÇB║BDB◙B B┤BB█oèBèB▬BzB¦B=B‼BrB▶BâB>B BÇB BtB‡BèB▬BzB¦B┐B BÇB BαB
┘B£B¦BΣB▒B B┘B║B¦BΣB½B BδB♠B┘BΣB¦BΣBúB BδB▪B B▶B B B B B B B B B B B B B B BB█
oo█oo█tt█B BLBoBaBdBeBrB BVB1B.B0B BUBnBaBbBlBeB BtBoB BrBeBaBdB BsBtBaBgBeB B2B
 BoBfB BtBhBeB BbBoBoBtB BpBrBoBcBeBsBsB BCBaBnBnBoBtB BcBoBnBt
```



```
Input > 4
Enter sectors to read
Input > 1
Enter message to search for
Input > ector
Reading next set of sectors
Sector 1 / 1

0000   6169 6c65 6420 746f 2072 6561 6420 6672 ailed to read fr
0010   6f6d 2064 6973 6b2e 0052 6561 6469 6e67 om disk._Reading
0020   206e 6578 7420 7365 7420 6f66 2073 6563  next set of sec
0030   746f 7273 0045 6e74 6572 2073 7461 7274 tors_Enter start
0040   696e 6720 7365 6374 6f72 0045 6e74 6572 ing sector_Enter
0050   2073 6563 746f 7273 2074 6f20 7265 6164  sectors to read
0060   0045 6e74 6572 206d 6573 7361 6765 2074 _Enter message t
0070   6f20 7365 6172 6368 2066 6f72 0049 6e70 o search for_Inp
0080   7574 203e 2000 5072 6573 7320 6120 6b65 ut > _Press a ke
0090   7920 746f 2063 6f6e 7469 6e75 652e 2e2e y to continue...
00a0   0053 6563 746f 7220 0020 2f20 0045 4e44 _Sector _ / _END
00b0   204f 4620 5052 4f47 5241 4d00 0000 0000  OF PROGRAM_____
00c0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00d0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00e0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
00f0   0000 0000 0000 0000 0000 0000 0000 0000 ________________
Press a key to continue...
```

After that, the search functionality worked as intended and I just needed to tidy up what I had. I found that a lot of the time I was saving all registers I used with every function which felt unneeded. As I continued development, I settled to save bx, cx and si when using them and trashing any other registers I used as pretty much everything remained the same. Arguably I could also trash bx however I just stuck to this standard by the end of it.

I understand that the graphics mode can make the text scroll quite difficult to look at comfortably however I couldn't find a way to fix this or an alternative way.