Contents

[04/11/2023 – 4 hours 2](#_Toc151119286)

[06/11/2023 – 3 hours 2](#_Toc151119287)

[10/11/2023 –2h 14m 4](#_Toc151119288)

[11/11/2023 4h 6](#_Toc151119289)

[14/11/2023 - 4h 8](#_Toc151119290)

[15/11/2023 1h 10](#_Toc151119291)

[15/11/2023 50m 11](#_Toc151119292)

# 04/11/2023 – 4 hours

Today I started off by cloning the starting template for the assignment and getting qemu working on my personal PC. Once done, I then began attempting to use the int 16 bios interrupt. This is the interrupt used to retrieve a keyboard input from a user. At first I was having issues with this as I didn’t fully understand the documentation for the interrupt table. Figure 1 shows the documentation for get keystroke, in order to use this, I had to pass in the value `00` into the AH register. When trying this I got an error which stated that there was a junk “h” after the expression, this is because I had already prefixed the `00` with 0x which denotes that it is a hexadecimal value, so removing the H resolved the problems.

A close-up of a computer code

Description automatically generated

Figure – interrupt documentation for keyboard input

# 06/11/2023 – 3 hours

Figure 2 shows me being able to type the number of the sector to read, this is currently not being used but I am creating the functionality for it. The next stage is to also display the sector number in the message “Contents of sector”, to do this I will be temporarily storing the contents of al inside of the stack, this will allow me to reuse it after the other console logs have been ran.

A screenshot of a computer

Description automatically generated

Figure - keyboard input working

When attempting to use the pushw instruction I ran into an error **“Error: unsupported instruction `push'”,** this is because I was using the %al register which is not supported with this instruction. After reading the lecture slides for week 3 it stated that you can only push the whole register onto the stack, once I changed it to use %ax I no longer received the error.

**Another issue which I have ran into is with the order that I was pushing and popping. I was using the pushw instruction after I call cons\_write\_crlf so the value in %al was being overwritten by the special characters. To fix the problem I had to use the pushw instruction before I called cons\_write\_crlf in order to save the correct value (the keyboard input)**

**A screen shot of a computer

Description automatically generated**

Figure - bug showing the wrong value being popped onto the register

**Figure 4 displays the code used to output the contents of figure 5.**

**A screenshot of a computer program

Description automatically generated**

Figure - The final code for keyboard input and contents of sector message

Figure 5 displays the final output for the code above. The next stage is to start reading the contents off of the disk.

A black screen with white text

Description automatically generated

Figure - final output of keyboard input label

# 10/11/2023 –2h 14m

**Today I had a bug where if I had a label under my `contents\_of\_sector\_message` it would output that to screen when it shouldn’t. This can be seen in figure 6.**

**A screenshot of a computer

Description automatically generated**

Figure - bug showing extra label being print to screen

After much time debugging and trying different things, such as adding a terminator value to the end of the content of sector string (even though .string adds this automatically) I was able to fix the issue. However, I didn’t quite understand why this was happening, so I had a look into it some more and later found that because I was calling cons\_writeline without moving a value into %si there was no null terminator, this caused the function to loop go through the wrong memory and access strings which it was not supposed to, the code for this can be seen in figure 7.

A screenshot of a computer program

Description automatically generated

Figure – code

To fix the problem in figure 7 I modified the code to include an empty string, which would act as a null terminator to stop the cons\_writeline going into the wrong area of memory. I later realised that cons\_writeline was for outputting not just a new line, but also the value in %si, so it I could just call cons\_write\_crlf to output the newline instead of calling cons\_writeline, which was for a different purpose.

The next stage was to start reading the disk, the initial code for this gave me some problems as I was just calling cons\_write\_hex, without passing any information into the register. This can be seen in figure 8.

A screenshot of a computer

Description automatically generated

Figure - cons\_write\_hex not outputting any data

In order to fix this problem I had a look at the code for outputting hex values, the register which was being used was BX. To get some initial output I moved the bytes that were inside of the memory address “0xD000” into the BL register. I then got the output in figure 9.

A screen shot of a computer

Description automatically generated

Figure - values being read from disk

Figure 10 displays an issue which I am having with my code. Currently, after attempting to output the next set of data onto the screen after the initial value of EB I am getting the wrong data. It appears that I am incrementing the value instead of the index.

A computer screen with white text

Description automatically generated

Figure - wrong output displayed.

# 11/11/2023 4h

I working yesterday and didn’t manage to fix the issue I had. After coming back today I thought about how the cons\_write label worked, using SI to loop through each character in a string. I am looking into implementing this sort of method for looping over the values that have been read from disk.

Figure 11 shows my attempt at using the SI register. Before getting this output I verified that the first iteration of the code output the correct value of “EB”. However, when trying to implement my loop any value after the first iteration was completely wrong, and also the same every time. At first I was completely clueless as to why this was happening, but as I went through each line I remembered a previous issue which I had earlier in the project which caused my “contents of sector x” message to break. When calling cons\_write\_hex and cons\_write the SI register is overwritten with the value inside of BX (Figure 12). This means that on the next iteration of the loop the value would be completely wrong, since it had been overwritten. The fix for this issue was to push the contents of SI onto the stack whilst cons\_write\_hex was running.

A black and white background with white letters

Description automatically generated

Figure - wrong values being displayed

A screenshot of a computer program

Description automatically generated

Figure

Figure 13 displays the correct output on the screen, the idea using %SI worked. However, the 2nd value in the output is different to what it shows in HXD. The value should be 20 but it says 3B. After looking at HXD some more I found a search function. I put the value EB 3B inside of the box and it found the values (figure 14). After looking at the offset it became clear that this data was part of a different block, to fix this problem I edited the first block to read to `0` and then I got the correct output in my console. The output can be seen in figure 15.

A screenshot of a computer

Description automatically generated

Figure - correct output

A screenshot of a computer

Description automatically generated

Figure

A black screen with white text

Description automatically generated

Figure

# 14/11/2023 - 4h

Today I made a lot of progress which I am very happy with. I had quite a few bugs which I ran into, some of them took quite a long time to fix.

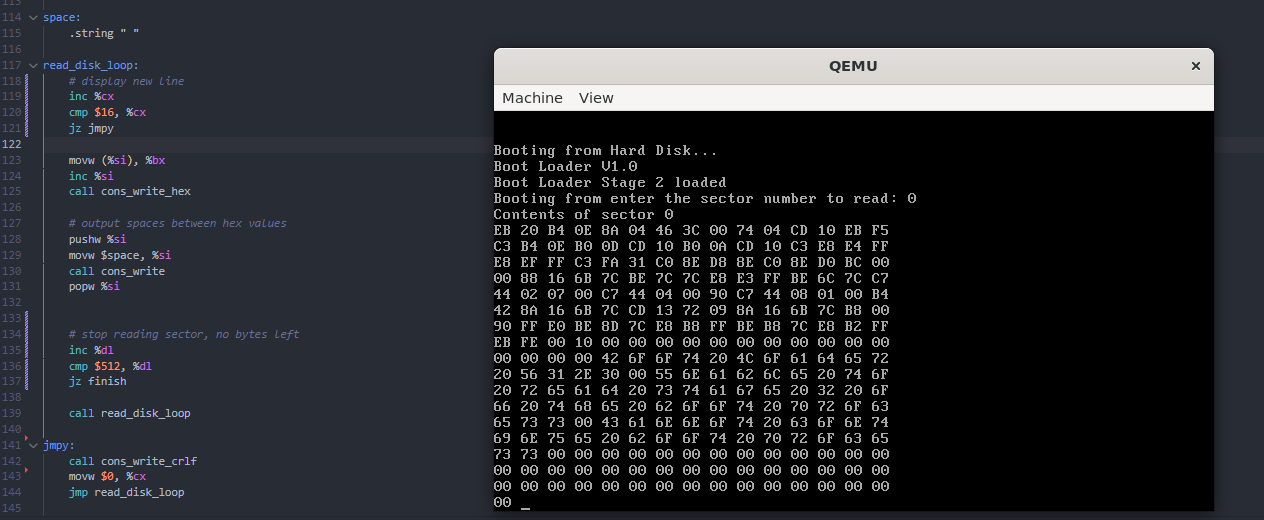
An issue which I had from the last coding session I did was that the output would stop when the data on the disk was the value “00”, this can be seen in figure 15. The cause of this was due to me comparing the value inside of %bx. My thought was if the bytes are 0 then there is nothing left to read, but I did not consider what would happen if the disk was empty. The fix for this issue was to use %dl as a counter to check how many bytes have been read, this can be seen in figure 16.

A screenshot of a computer

Description automatically generated

Figure

The next step I took was to ensure that only 16 bytes per line was being displayed, to do this I made use of the CX register to keep a count of how many bytes had been written to the screen. This can be seen in figure 17. Every iteration of the read\_disk\_loop that happens %cx is incremented. If the value of cx is 16 then it calls a label which writes a new line and resets CX back to 0. This is repeated until all bytes have been read.



Figure

A glaring issue with figure 17 is that an extra byte is read at the end of the output, after a long time of debugging I eventually found out that the problem was due to me trying to compare %dl with a value of 512, and since %dl is an 8bit register the maximum number that could be inside of it was 255, so the check to know weather or not it has finished was not being executed properly.

To fix this, instead of checking how many individual bytes have been written I used a smaller number of $16, this represented the number of lines that had been output onto the screen. The correct output can be seen in figure 18, which I verified by using HXD.

A screenshot of a computer

Description automatically generated

Figure

# 15/11/2023 1h

Showing keyboard input working

A screenshot of a computer

Description automatically generated

Figure

# 15/11/2023 50m

Today I began working on writing the offset into the sector onto the screen. Whilst working on this feature I first began by removing all of the code in read\_disk\_loop so I could easily verify the output of my offsets, this made it easier for me to focus on the code without it being bloated or getting mixed onto different lines. To prevent once I had verified my approach worked, I began trying to implement it.

The steps for proving my plan was valid was to copy cons\_write\_hex and duplicate it, this was because the values in cx to determine how the hex values get print to the screen had to be different for printing the offset. I realize that duplicating an entire label is not ideal, so once I have it working properly I will likely be requiring the user to change the value of cx before calling the cons\_write\_hex label.

A screen shot of a computer screen

Description automatically generated

When trying to output these values once I had brought the original code back, I was getting incorrect values (figure 20) because the cons\_write\_hex label prints the entirety of bx, so when I was moving DL into BL there were still values in bh. To fix this issue I ended up splitting up my character counter and line counter into CH and CL instead of using the entire register for each, this meant that I was able to use the entirety of DX to store the offset value and move it into BX and get the correct output seen in figure 21.

A screenshot of a computer

Description automatically generated

Figure

Figure 1 shows the offset into the sector being displayed; however, it is slightly wrong. It displays the offset at the end of the line, and it also displays the wrong value for line one, it should start from 0.

A screenshot of a computer

Description automatically generated

Figure

# 16/11/2023 2h

During this time frame I attempted to change the first value of the offset, so it started from 0000 instead of 0010. As well as move the position of the offsets so they are displayed at the start of the line. This was a relatively simple task although it took me a long time due to me trying to figure out how to use the div instruction, after much time attempting to use it so I could check if the value in cl was a multiple of 16 which meant a new line would be printed, I could use this information to tell the offset to be printed to the start of the line. However, I realised that I don’t necessarily have to do it this way, if I had thought about what I was doing a little bit more I would have realised a lot sooner that I could’ve just reordered the code so that the offset is output before the hex values. Figure 21 displays the offset being correct, however the last offset is printed on line 16. To fix this I had to add a cmp statement which checked the current line number to see if the current line was the last line which outputs any hex codes.

    cmp $15, %cl

    jz last\_line\_of\_sector

A computer screen shot of a number

Description automatically generated

Figure

# 17/11/2023

My next task to complete is the part “Press a key to continue reading sector” message. To do this I will use the label I have previously made get\_keyboard\_input. However, since I have begun the project more and more bloat has been added to this label and now its name does not properly represent what it does, not only does it get keyboard input but it also outputs contents onto the screen, because of this it makes reusing this label quite difficult, this is why I will be refactoring some code so that this label has a single purpose.

Attempt at ascii output

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

WOOOOO

A screenshot of a computer

Description automatically generated

Finished ascii output

A screenshot of a computer

Description automatically generated

Continue reading sector…

A screenshot of a computer

Description automatically generated