

Summer 2021

California State University, Northridge

Department of Electrical & Computer Engineering

MINI project3

July 12, 2021

Ryan Mark

```

1 ; Ryan mark
2 ; mini project 3
3         AREA mini3, CODE, READONLY
4 rows           EQU 16 ; amount of rows in image
5 collum          EQU 16 ; amount of collums in image
6 storage         EQU 0x40000000 ;adress of the unfiltered image
7 filter_image    EQU 0x40000100 ;the adres of the image that is going to be filtered
8 Histo1          EQU 0x40000200 ;stores data for the histogram of unfiltered image
9 Histo2          EQU 0x40000300 ; stores data for histogram of filtered image
10 intensity       EQU 0x40000400 ; store the the list inntenitrys found in unfiltered image
11 filt_intensity EQU 0x40000500 ; store the list of intensity found in filtered immage
12
13         ENTRY
14         MOV r2 ,#collum ;counter for columns
15         MOV r3,#storage
16 Image          DCD row1,row2,row3,row4,row5,row6,row7,row8,row9,row10,row11,row12,row13,row14,row15,
17 row16; array of all the rows
18         ALIGN
19
20         LDR r0,= Image ;loads adress of the 2d image
21
22 ;-----2d map to
23 1d-----1d-----
24 ONE_D          LDR r4,[r0],#4
25             MOV r1,#rows ; counter for elements with thin a row
26 ELEMENT        LDRB r5,[r4],#1
27             STRB r5,[r3],#1
28             SUBS r1,r1,#1; counter goes down
29             BNE ELEMENT
30             SUBS r2,r2,#1
31             BNE ONE_D
32 ;-----1d to 2d
33 map-----map-
34
35             LDR r0,= filter_image;image that will be filtered later on
36             MOV r2, #collum;total number of number in the 16x16 matrix
37             MOV r3,#storage
38
39 COPY           MOV r1,#256 ; counter for elements
40             LDRB r4,[r3],#1;
41             STRB r4,[r0],#1; create duplitcae of input that will be later fileter as the output
42             SUBS r1,r1,#1
43             BNE COPY
44
45 ;-----histogram for unfiltered
46 image-----image-
47
48             LDR r0,=Histo1; will contain data for 1st histogram
49             MOV r1,#0 ; counter used to co mpare vaules in the image
50             MOV r4,#256 ;hiogram loop counter
51             LDR r7, =intensity
52 HISTO          MOV r2, #0; counter for how many of what pixel intesity is in the image in
53             MOV r3, #256 ;total number of element in the 16x16 matrix
54             LDR r5, =storage; get adress of the one dimensional array used to compare and also
55             reset r5 so it can be used again
56 ;-----array number search-----
57 search          LDRB r6,[r5],#1
58             CMP r6,r1 ; compare the value of the element to the number that is being searched for
59             ADDEQ r2,r2,#1; add to counter if one of the element is equal to the number being
60             compared too
61             SUBS r3,r3,#1
62             BNE search
63 ;-----end of array number search loop-----
64             CMP r2,#0 ;check if r2 has any number of a specific pixel intesity
65             STRBNNE r2 ,[r0],#1; stores counter of the number of a sepcific pixel intesity if
66             couter is not zero
67             CMP r2,#0
68             STRBNE r1,[r7],#1 ; store the intesity that exists in the image
69             ADD r1,r1,#1; set the next number that going to be searched and comapered with
70             SUBS r4,r4,#1 ; histogram counter goes down
71             BNE HISTO
72 ;-----image
73 filter-----filter-
74             LDR r0, =Image ; the original image

```

```

65           LDR r1, =filter_image ; the image that will be filtered
66           ADD r1,r1,#17 ; goes to the address row 2 column 2 which will be the first pixel filtered
67           MOV r12,#14; counter for rows when excluding row 1 and 16
68   ;----- image filter
69   loop-----
70   image_filter
71   change r0           LDR r3,[r0,#4]; goes to row 2 by getting the address of row2 pre indexing used to not
72   change r0           LDR r4,[r0,#8]; goes to row 3 by getting the address of row3 pre indexing used to not
73   change r0           LDR r2,[r0],#4 ; goes to row 1 by getting the address of row post indexing used to
74   change r0 by adding 4 to address to shift the starting point down by 1 row for the next cycle
75   ;-----pixel calculation assuming 3x3 matrix use for
76   calculation-----
77   pixel_filter        MOV r9,#0; value for filter of a pixel
78           LDRB r7,[r2,#1]; row 1 column 2 value
79           LDRB r8,[r2,#2]; row 1 column 3 value a
80           LDRB r6,[r2],#1; row 1 column 1 value also adds one to the address so we go through all
81   columns
82   ; we do not multiply (row1,column1) since it being multiplied by one
83   MOV r7,r7,LSL #1; multiply (row1,column 2) by 2
84   ; we do not multiply (row1,column3) since it being multiplied by one
85   ADD r7 ,r6,r7; (row1,column1) + (row1,column2)
86   ADD r8,r7,r8      ; (row1,column1) + (row1,column2)+ (row1, column3)
87   ADD r9,r9,r8      ;0 +(row1,column1) + (row1,column2)+ (row1, column3)
88
89   LDRB r7,[r3,#1]; row 2 column 2 value
90   LDRB r8,[r3,#2]; row 2 column 3 value
91   LDRB r6,[r3],#1; row 2 column 1 value also adds one to the address so we go through all
92   columns
93   MOV r6,r6,LSL #1 ;multiply (row2,column1) by 2
94   MOV r7,r7,LSL #2; multiply (row2,column2) by 4
95   MOV r8,r8,LSL #1 ; multiply ((row2,column2) by 2
96
97   ADD r7 ,r6,r7; (row2,column1) + (row2,column2)
98   ADD r8,r7,r8      ; (row2,column1) + (row2,column2)+ (row2, column3)
99   ADD r9,r9,r8      ; (row1,column1) + (row1,column2)+ (row1, column3)+(row2, column1) +
100  (row2, column2)+ (row2, column3)
101
102  LDRB r7,[r4,#1]; row 3 column 2 value
103  LDRB r8,[r4,#2]; row 3 column 3 value a
104  LDRB r6,[r4],#1; row 3 column1 value also adds one to the address so we go through all
105 columns
106
107  ; we do not multiply (row3,column1) since it being multiplied by one
108  MOV r7,r7,LSL #1; multiply by 2
109  ; we do not multiply (row3,column3) since it being multiplied by one
110
111  ADD r7 ,r6,r7; (row3,column1) + (row3,column2)
112  ADD r8,r7,r8      ;(row3,column1) + (row3,column2)+ (row3, column3)
113  ADD r9,r9,r8      ;(row1,column1) + (row1,column2)+ (row1, column3)+(row2, column1) +
114  (row2, column2)+ (row2, column3) +(row3, column1) + (row3, column2)+ (row3, column3)
115
116  MOV r9,r9,LSR #4 ; divide by 16 by shifting binary form by 4 to the right
117  STRB r9,[r1],#1; store new filter value in designated destination and shifts the pixel
118  filter by one
119  SUBS r13,r13,#1
120  BNE pixel_filter
121 ;-----end of pixel filter loop-----
122  ADD r1,r1,#2; makes it so r1 has the address of the next row while also making sure it
123  starts at column 2 of the next row
124  SUBS r12,r12,#1 ; row counter goes down by 1
125  BNE image_filter
126 ;----- end of image filter loop-----
127
128 ;-----histogram for filtered

```

image-----

```

125          LDR r0,=Histo2; will contain data for 1st histogram
126          MOV r1,#0 ; counter used to compare values in the image
127          MOV r4,#256 ; histogram loop counter
128          LDR r7, =filt_intensity
129 HISTO_FILTER    MOV r2, #0; counter for how many of what pixel intensity is in the image in
130          MOV r3, #256 ;total number of element in the 16x16 matrix
131          LDR r5, =filter_image; get address of the one dimensional array used to compare
132          ;-----array number search-----
133          ;-----array number search-----
134 search_filter   LDRB r6,[r5],#1
135          CMP r6,r1
136          ADDEQ r2,r2,#1; add to counter if one of the element is equal to the number being
137          compared too
138          SUBS r3,r3,#1
139          BNE search_filter
140          ;-----end of array number search loop-----
141          CMP r2,#0 ;check if r2 has any number of a specific pixel intensity
142          STRBNE r2,[r0],#1; stores counter of the number of a specific pixel intensity if
143          counter is not zero
144          CMP r2,#0
145          STRBNE r1,[r7],#1 ; store the intensity that exists in the image
146          ADD r1,r1,#1; set the next number that going to be searched and compared with
147          SUBS r4,r4,#1 ; histogram counter goes down
148          BNE HISTO_FILTER
149
150 stop           B stop
151 ;the image
152 row1          DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
153 row2          DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
154 row3          DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
155 row4          DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
156 row5          DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
157 row6          DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
158 row7          DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
159 row8          DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
160 row9          DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
161 row10         DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
162 row11         DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
163 row12         DCB 0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255
164 row13         DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
165 row14         DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
166 row15         DCB 255,255,255,255,0,0,0,0,255,255,255,255,0,0,0,0
167
168
169 END

```

START of the code

The screenshot shows the uVision IDE interface with the following windows:

- Registers**: Shows the current register values for R0 through R15, CPSR, and SPSR.
- Disassembly**: Displays the assembly code for the mini3.s project. The code includes declarations for parameters (rows, columns, storage), histograms (Histol, Histol2), intensity lists (intensity, filt_intensity), and arrays (Image). It also includes memory allocations (DCD) and loads (LDR) for the 2D image.
- Memory**: Shows the memory dump starting at address 0x40000000, where all memory is currently filled with zeros.
- Command**: Displays the command-line interface output, including the command to run the code and the loaded file path.

```

1: Ryan mark
2 : mini project 3
3          AREA mini3, CODE, READONLY
4 rows      EQU 16 ; amount of rows in image
5 column    EQU 16 ; amount of columns in image
6 storage   EQU 0x40000000 ;address of the unfiltered image
7 filter_image EQU 0x40000100 ;the adres of the image that is going to be filtered
8 Histol    EQU 0x40000200 ;stores data for the histogram of unfiltered image
9 Histol2   EQU 0x40000300 ; stores data for histogram of filtered image
10 intensity EQU 0x40000400 ; store the the list intninteniy found in unfiltered image
11 filt_intensity EQU 0x40000500 ; store the list of intensity found in filtered imgage
12
13          ENTRY
14         MOV r2 ,#column ;counter for columns
15         MOV r3,#storage
16 Image     DCD row1,row2,row3,row4,row5,row6,row7,row8,row9,row10,row11,row12,row13,row14,row15,row16; array of all the rows
17         ALIGN
18
19         LDR r0,= Image ;loads adress of the 2d image

```

Figure 1 proof code is able to start

TASK 1

Store the image in the memory. You need to map the two dimensional image data to one dimensional array. You must be able to map the reverse process to find the two dimensional data from one dimensional array. What is the mapping formula for each case for this image clip? Can you present the mapping formula for a general case of $M \times N$ image?

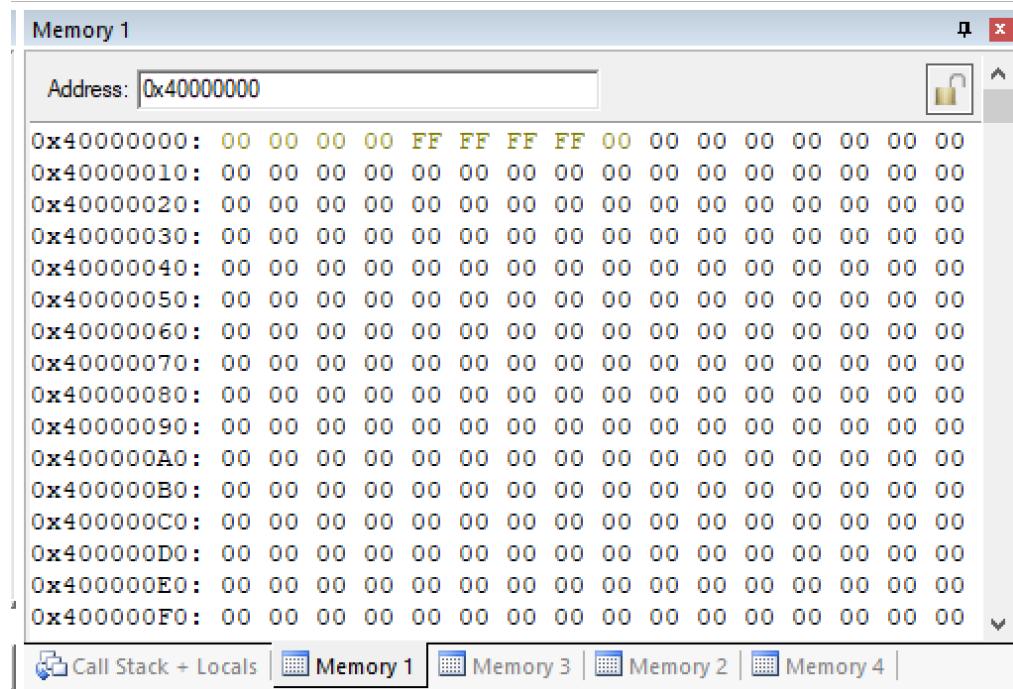
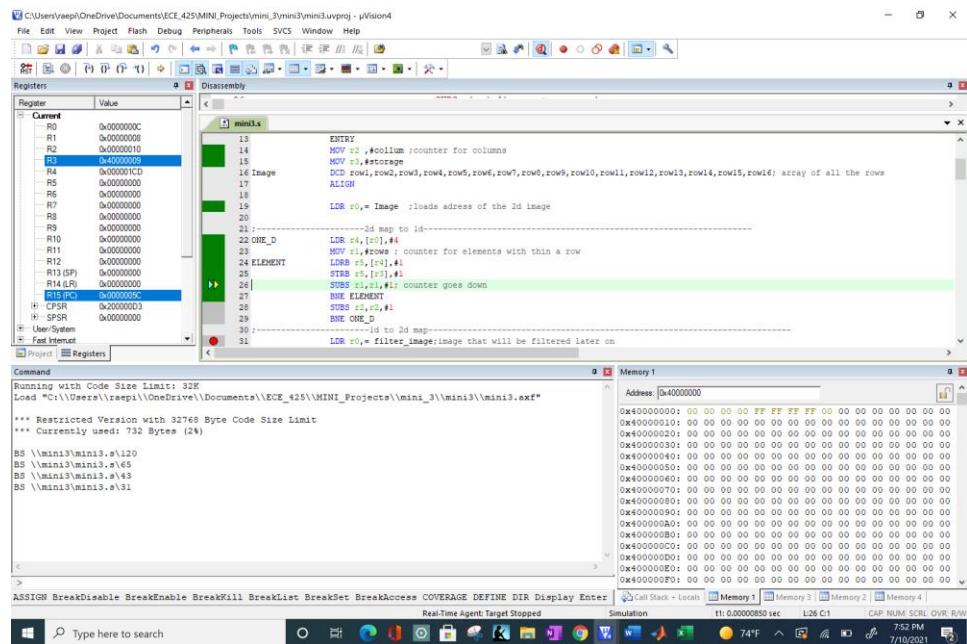


Figure 2: image in the process of being stored in address 0x40000000

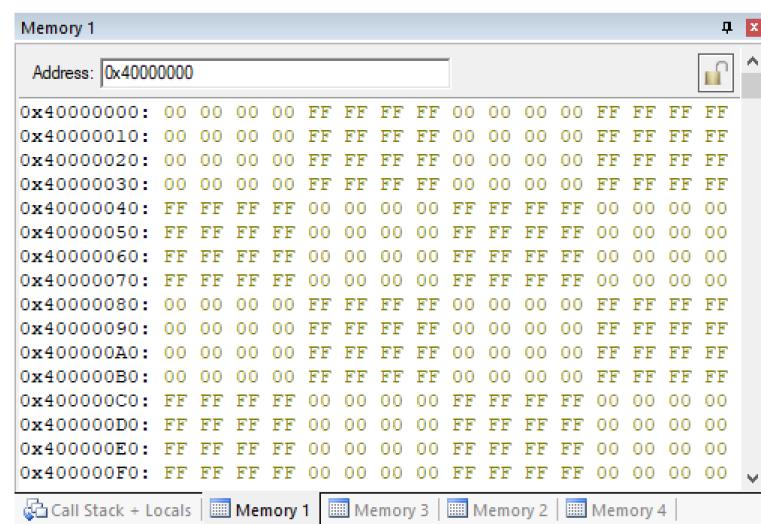
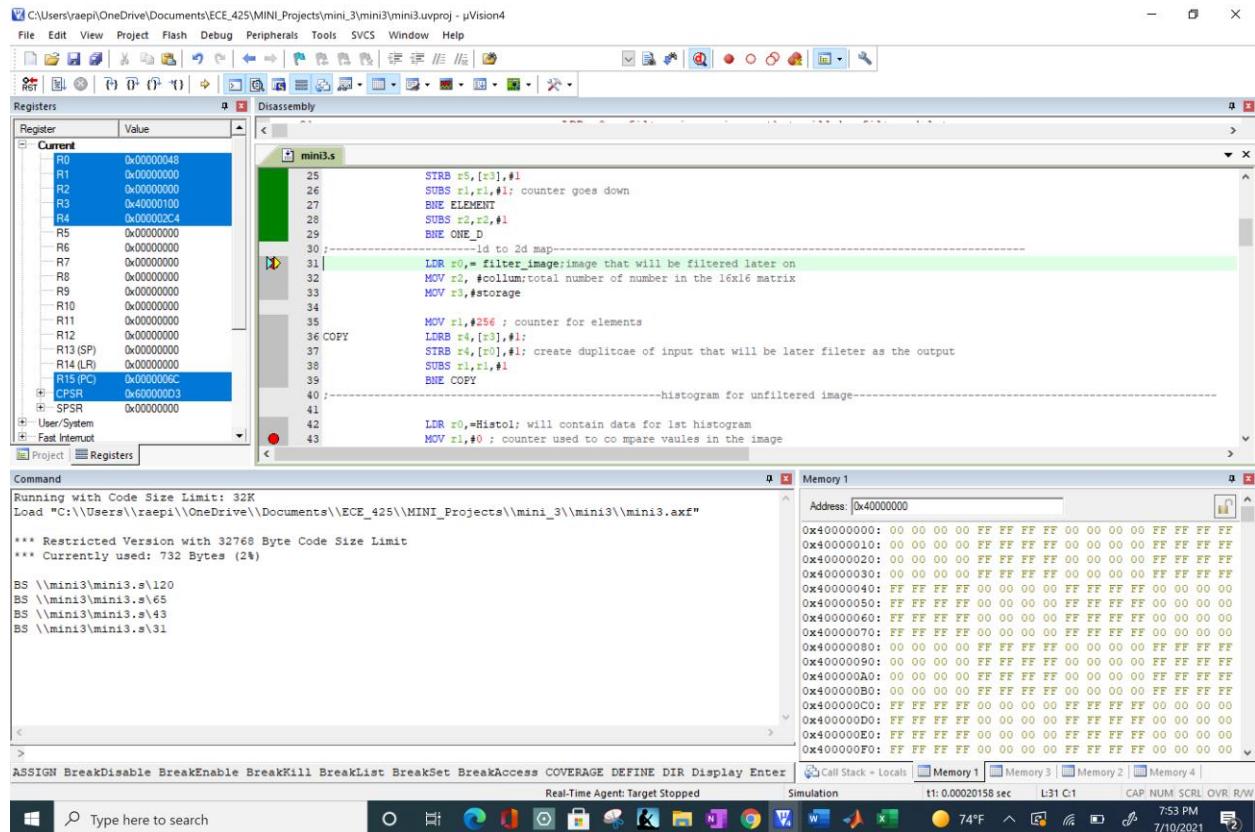
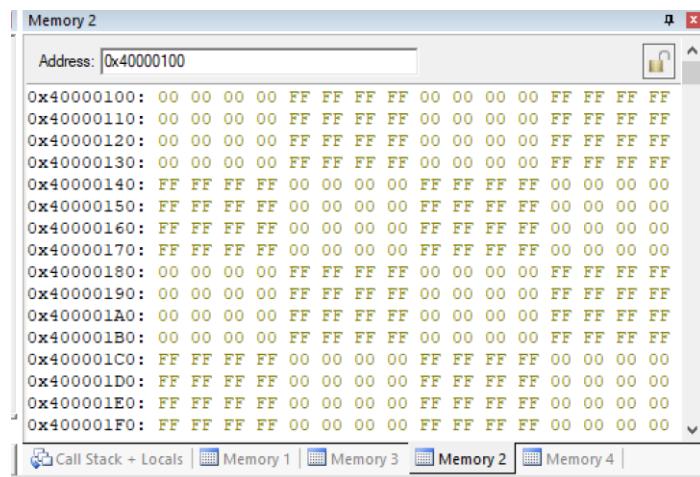
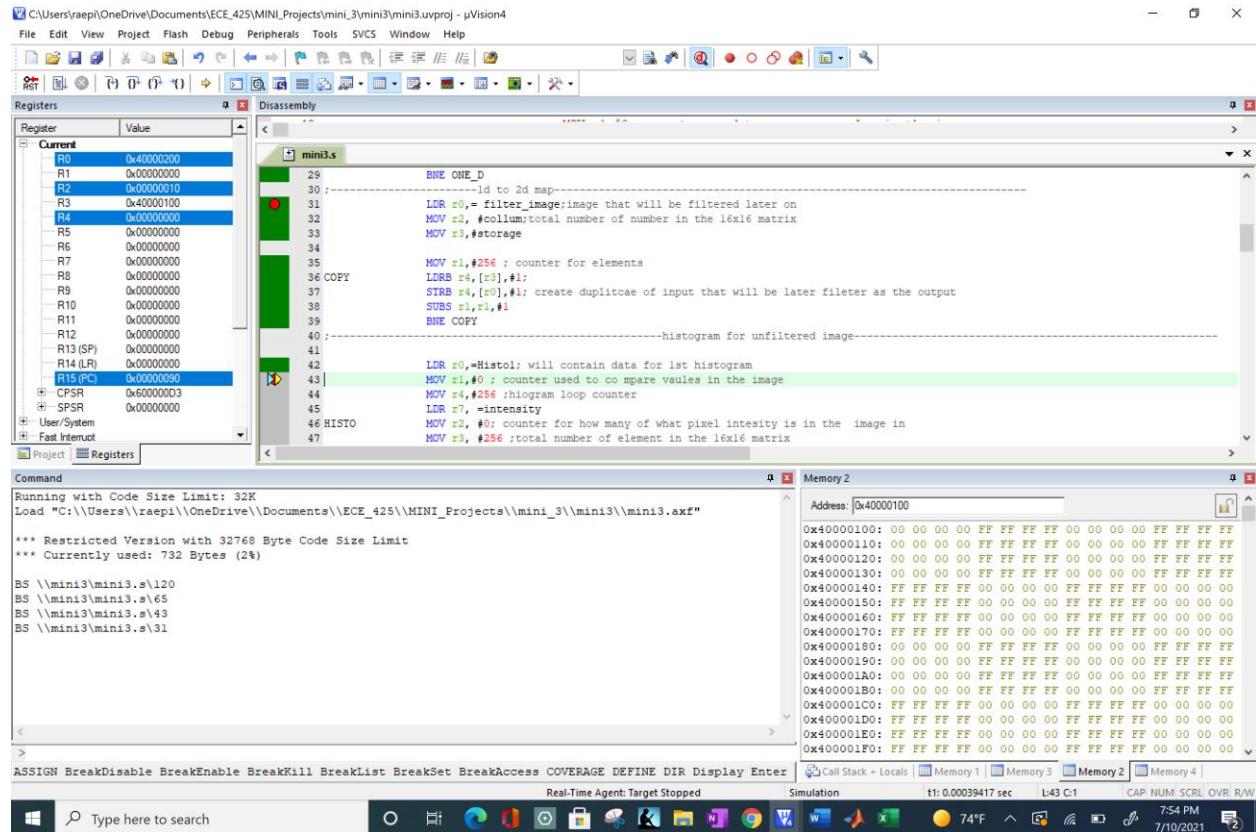


Figure 3 full image written into the address 0x40000000

For the mapping formula I take I make a nested loop to create the 1d array the main loop goes through the rows starting with row 1 and the sub loop goes through each element within the and the nested loop keeps cycling until it has gone through all 16 rows. For example on the first cycle we go through row one the sub loops cycles 16 times for each column to get all 16 element for a row.

Making a copy of the image into address 0x40000100 which will be the image I eventually filter



Task 2

Write an ARM assembly program to calculate the image histogram and use Excel to provide the histogram graph. Your histogram should display the pixel intensity values on x axis and the number of pixels with each intensity value on the y axis. For the given image, this is a trivial operation as there are only two intensities: 0 and 255. Your program should read one input array (image data) and output two arrays: pixel values (intensity values) and their corresponding number of pixels with that intensity value. For the input image, this would be 0 and 255 on the x axis and the number of pixels for each value (128 and 128) on the y axis

Basically, what I did to get the histogram data is I made a nested loop one to go through all 256 types of pixel intensity's and the sub loop goes through the image to see how many of that pixel intensity exists within the image. If the intensity is found that intensity that exist is recorded into address 0x40000400. The amount of pixel that have the found intensity is recorded into address 0x40000200. Due to the nature of my method of getting the histogram it should be able to work with any image.

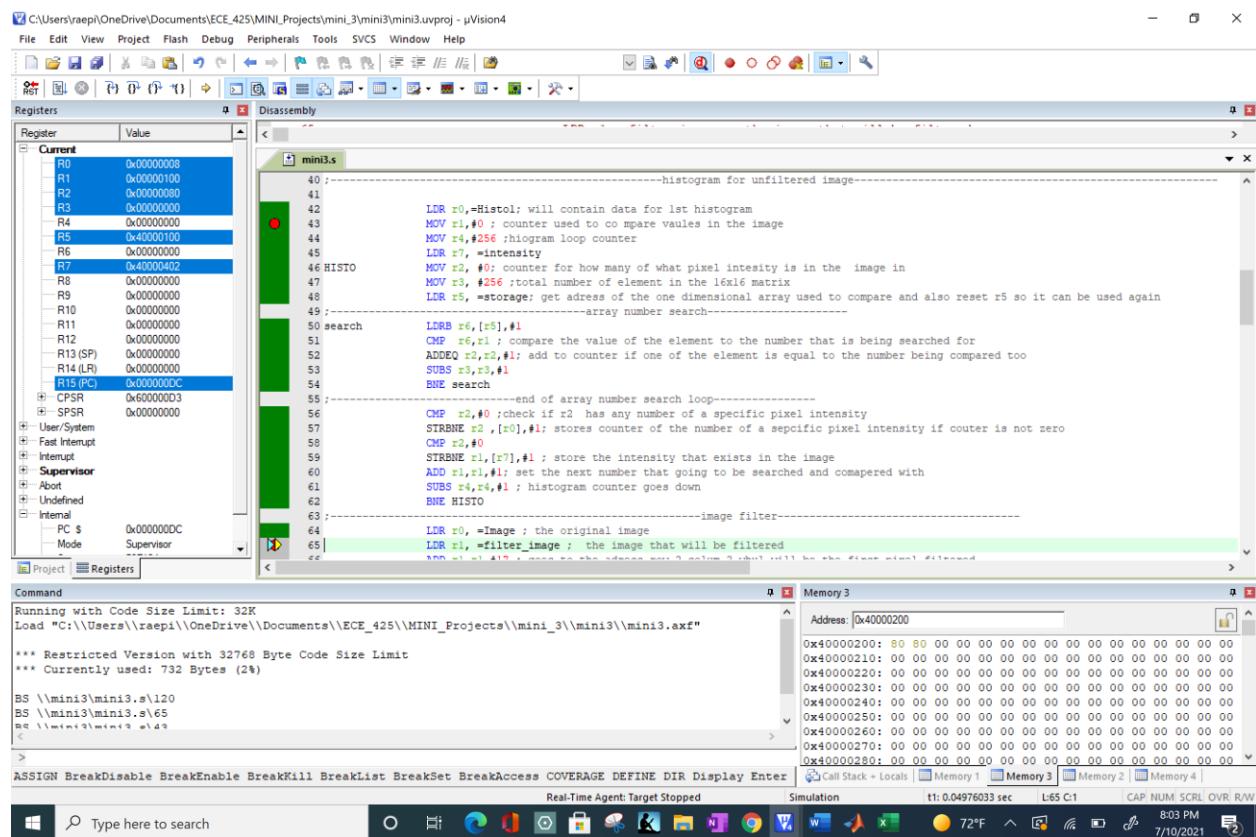


Figure 2.1 : Code for getting histogram data

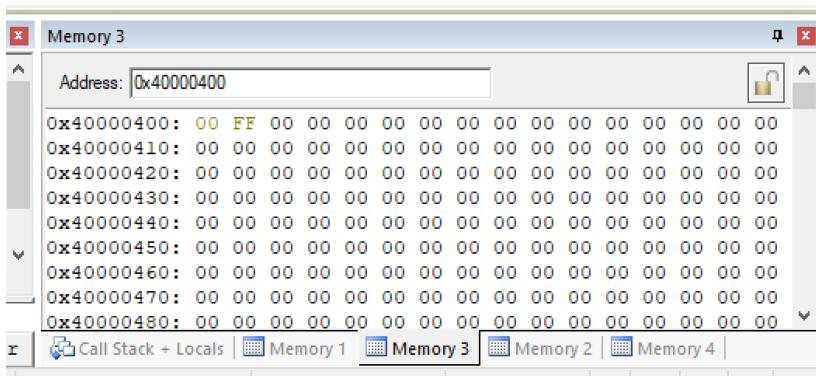


Figure 2.2 : intensity's found in unfiltered image stored in address 0x40000400. The intensity 0 and 255 exist for the u filtered image.

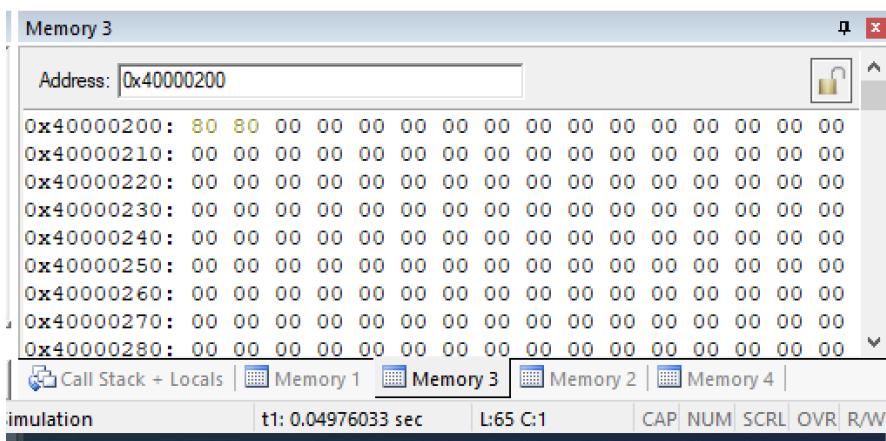


Figure 2.3 :Number of the found intensity for unfiltered image stored in address 0x40000200, Show 128 (80in hexadecimals) pixel had intensity 0 were found and 128 (80 in hexadecimals) pixel had intensity 255 were found

TASK 3

3. Write an ARM assembly program to apply image averaging filter to the original given image. Your program should read all image pixel values and apply the averaging filter to find the new values. Your program should process all image data in a loop to produce filtered image. Store the filtered image at different address

This part of the project was probly the most challenging part. I was able to complete this task by using the original image to help make calculation for every filtered pixel and store them into the duplicate image I made earlier. That way the filtered image is in a completely different address than the original. I crated the filter image using a nested loop I made use of pre and post indexing so I could access 3 rows at a time while also only shifting down 1 row as oppose to 3 rows that way everything get filtered in the end with the exception of row 1 and 16. For example the 1st cyle uses rows 1, 2,3 to filter row 2 and for the 2nd cycle it will use rows 2,3,4 to filter row 3. The pixel calculation loop goes through all column for

the row that being filter and like the main loop it makes use of pre and post indexing to get 3 element in a single row while only shifting by 1 after getting 3 elements.

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current register values, including R0 through R15, CPSR, and SPSR.
- Disassembly**: Displays the assembly code for the `mini3.s` file. The code implements a 3x3 matrix multiplication for filtering. It uses registers r5, r6, r7, r8, and r9 to store intermediate values. The assembly code includes comments explaining the calculation steps and memory access patterns.
- Command**: Shows the command-line interface with build logs and memory dump options.
- Memory 2**: A memory dump window showing memory starting at address 0x40000100 filled with FF (hexadecimal).

```

75; -----pixel calculation assuming 3x3 matrix use for calculation-----
76 pixel_filter    MOV r9,#0; value for filter of a pixel
77    LDRB r7,[r2,#1]; row 1 column 2 value
78    LDRB r8,[r2,#2]; row 1 column 3 value a
79    LDRB r6,[r2],#1; row 1 column 1 value also adds one to the address so we go through all columns
80
81; we do not multiply (row1,column1) since it being multiplied by one
82    MOV r7,r7,LSL #1; multiply (row1,column 2) by 2
83; we do not multiply (row1,column3) since it being multiplied by one
84
85    ADD r7 ,r6,r7; (row1,column1) + (row1,column2)
86    ADD r8,r7,r8 ; (row1,column1) + (row1,column2)+(row1, column3)
87    ADD r9,r8,r8 ;0 +(row1,column1) + (row1,column2)+(row1, column3)
88
89    LDRB r7,[r3,#1]; row 2 column 2 value
90    LDRB r8,[r3,#2]; row 2 column 3 value
91    LDRB r6,[r3],#1; row 2 column 1 value also adds one to the address so we go through all columns
92
93    MOV r6,r6,LSL #1 ;multiply (row2,column1) by 2

```

Figure 3.1 : calculating filtered pixels for the 2nd row

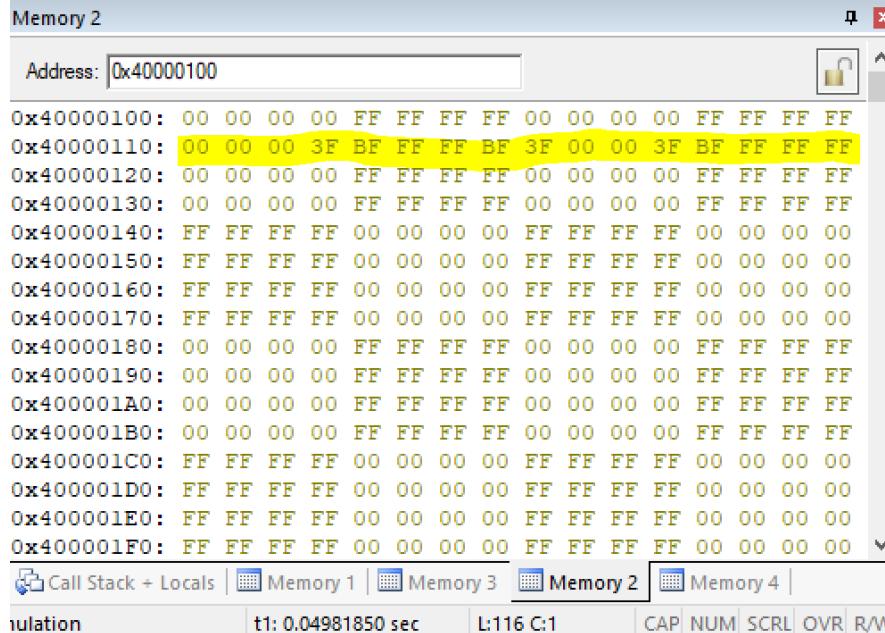
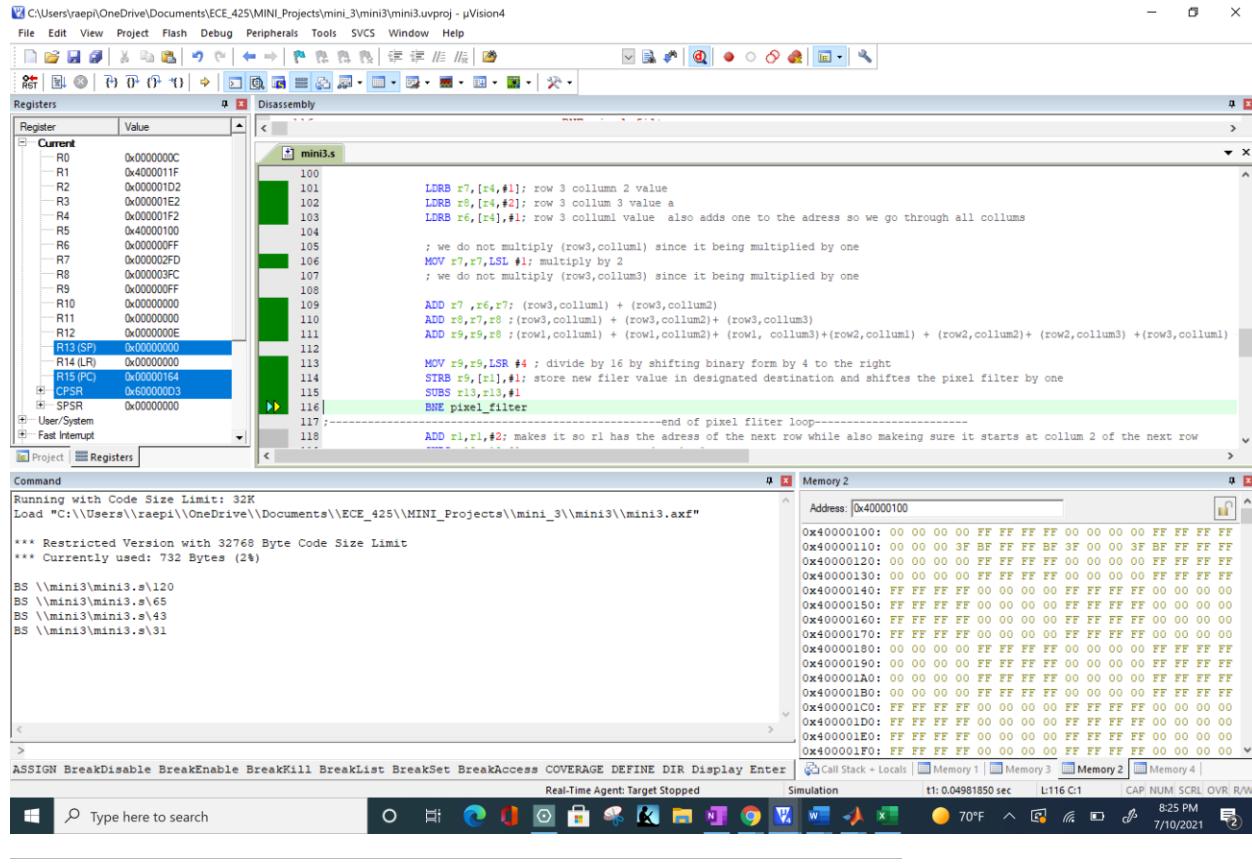


Figure 3.2 : row 2 has been filtered

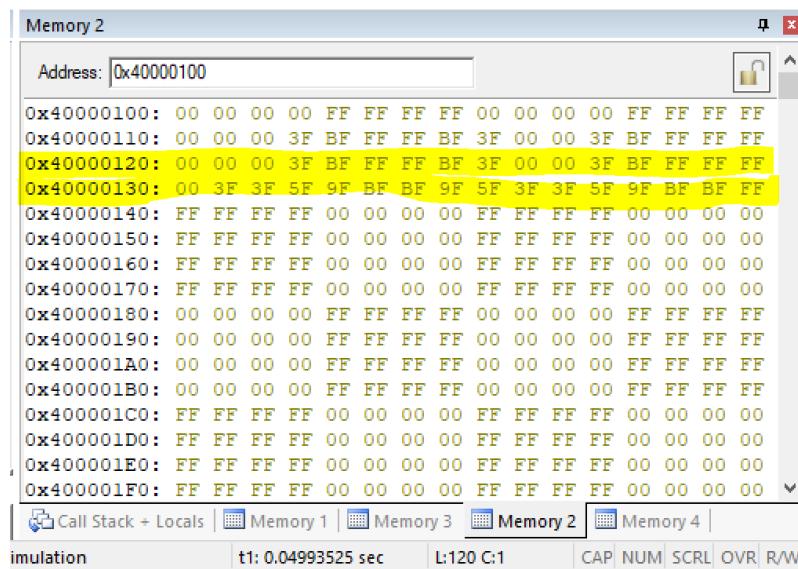
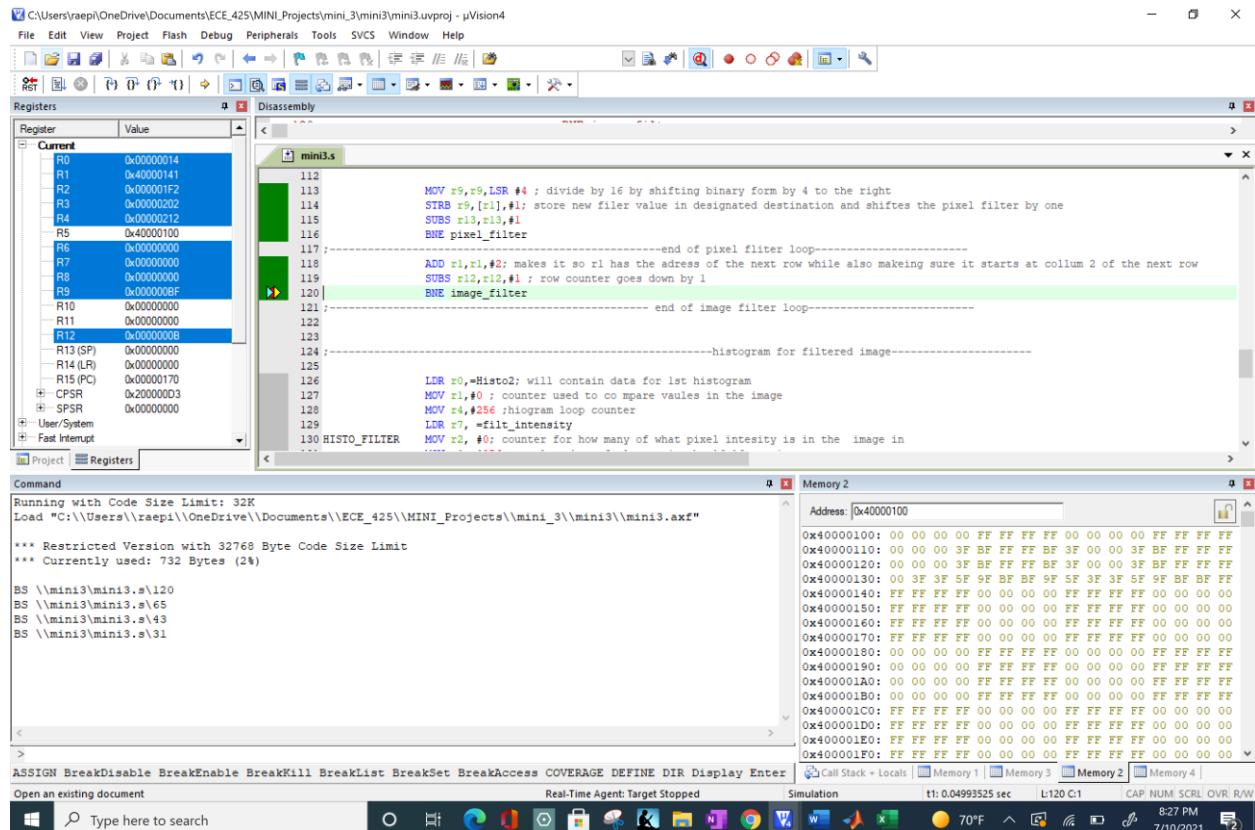


Figure 3.3 row 3 and 4 filtered

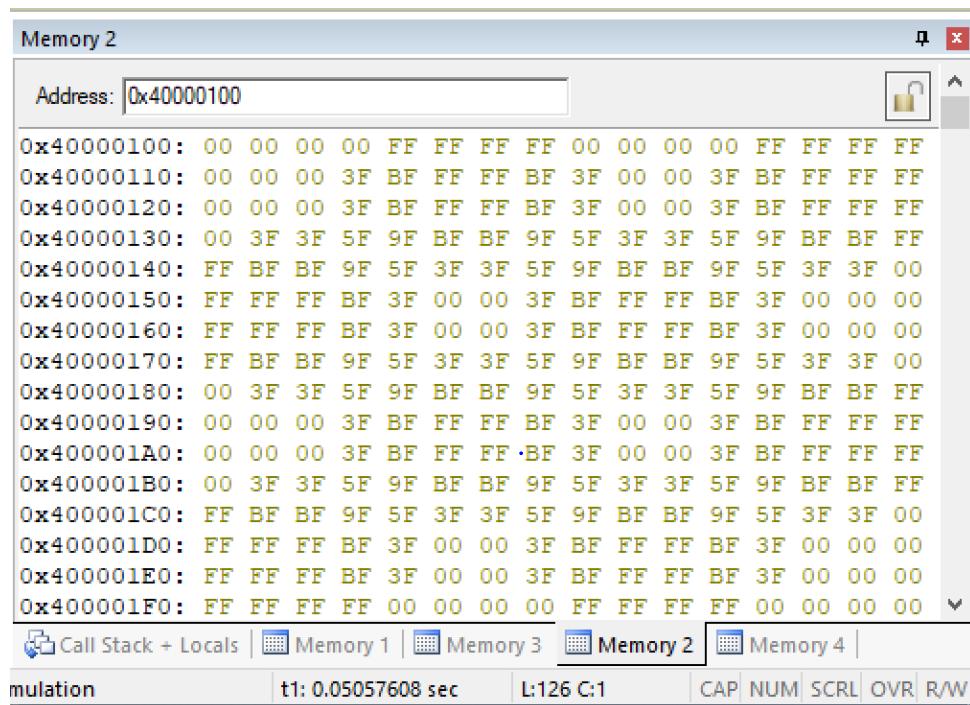
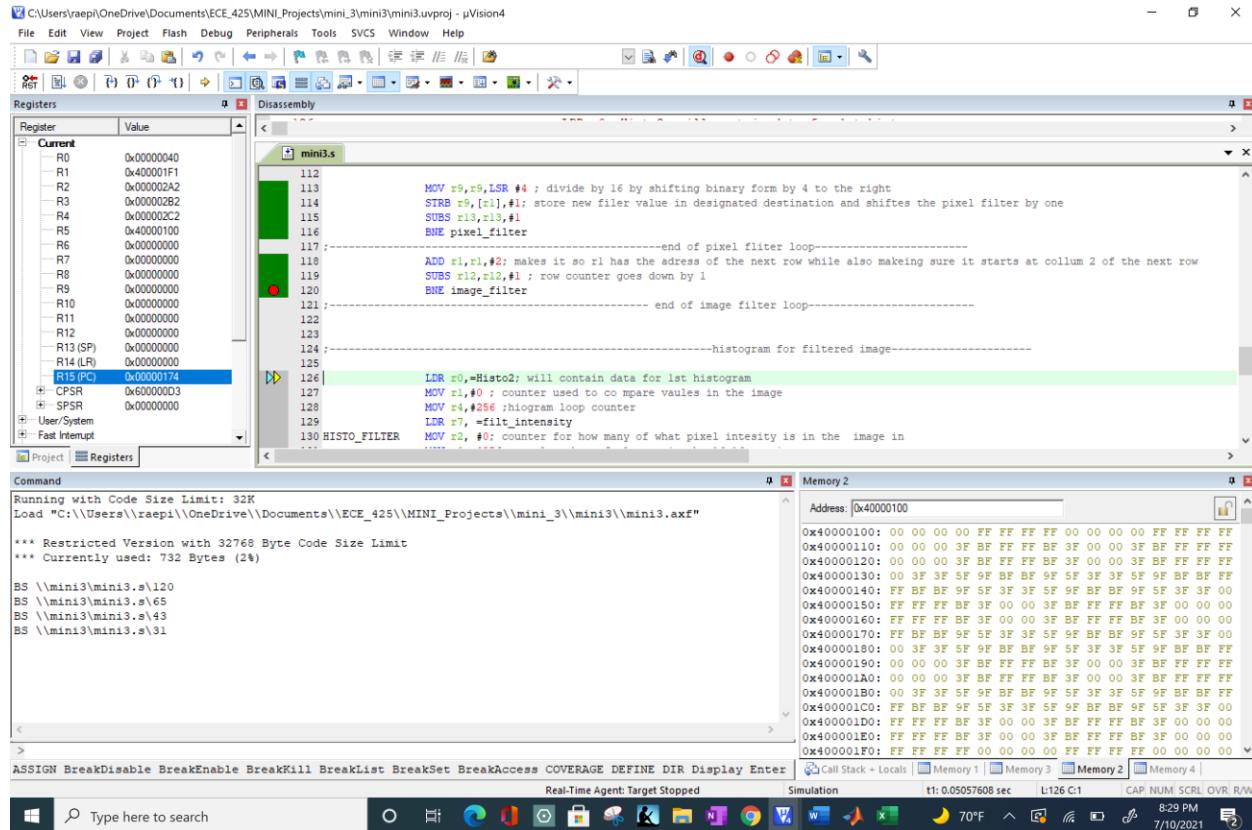


Figure 3.4 completed filtered image

Before image was filtered in address 0x40000000

Memory 1

Address:	0x40000000
0x40000000:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x40000010:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x40000020:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x40000030:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x40000040:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x40000050:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x40000060:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x40000070:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x40000080:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x40000090:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x400000A0:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x400000B0:	00 00 00 00 FF FF FF FF 00 00 00 00 00 FF FF FF FF
0x400000C0:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x400000D0:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x400000E0:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00
0x400000F0:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00

Call Stack + Locals | Memory 1 | Memory 2 | Memory 3 | Memory 4 |

After image was filtered in address 0x40000100

Memory 2

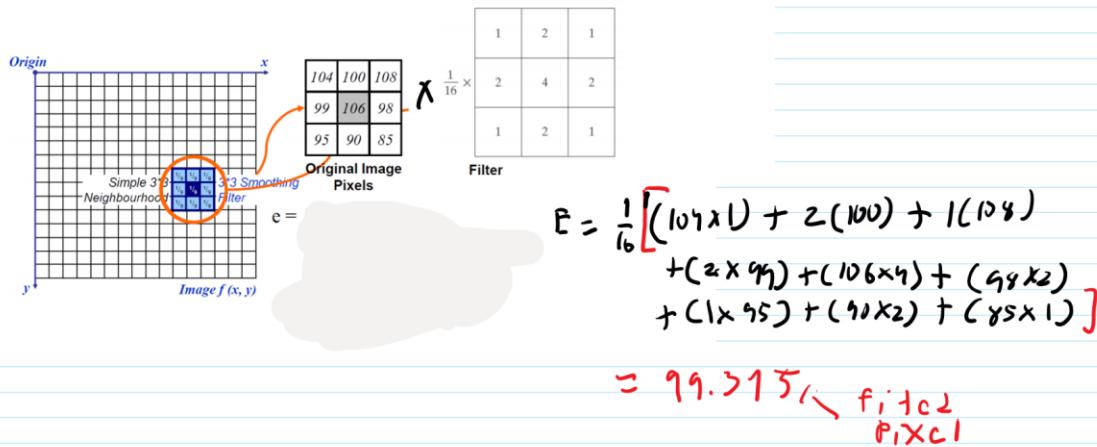
Address:	0x40000100
0x40000100:	00 00 00 00 FF FF FF FF 00 00 00 00 FF FF FF FF
0x40000110:	00 00 00 3F BF FF FF BF 3F 00 00 3F BF FF FF FF
0x40000120:	00 00 00 3F BF FF FF BF 3F 00 00 3F BF FF FF FF
0x40000130:	00 3F 3F 5F 9F BF BF 9F 5F 3F 5F 9F BF BF FF
0x40000140:	FF BF BF 9F 5F 3F 3F 5F 9F BF BF 9F 5F 3F 3F 00
0x40000150:	FF FF FF BF 3F 00 00 3F BF FF FF BF 3F 00 00 00
0x40000160:	FF FF FF BF 3F 00 00 3F BF FF FF BF 3F 00 00 00
0x40000170:	FF BF BF 9F 5F 3F 5F 9F BF BF 9F 5F 3F 3F 00
0x40000180:	00 3F 3F 5F 9F BF BF 9F 5F 3F 5F 9F BF BF FF
0x40000190:	00 00 00 3F BF FF FF BF 3F 00 00 3F BF FF FF FF
0x400001A0:	00 00 00 3F BF FF FF BF 3F 00 00 3F BF FF FF FF
0x400001B0:	00 3F 3F 5F 9F BF BF 9F 5F 3F 5F 9F BF BF FF
0x400001C0:	FF BF BF 9F 5F 3F 3F 5F 9F BF BF 9F 5F 3F 3F 00
0x400001D0:	FF FF FF BF 3F 00 00 3F BF FF FF BF 3F 00 00 00
0x400001E0:	FF FF FF BF 3F 00 00 3F BF FF FF BF 3F 00 00 00
0x400001F0:	FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00

Call Stack + Locals | Memory 1 | Memory 3 | **Memory 2** | Memory 4 |

Task 4

What did happen after applying the filter image? Explain your answer based on the new matrix values

Each filter image I calculating by using a 3x3 image of the orginal image the. The center of it is what being filtered in order to get it we multiply it by the corespong element of the kernel image which is [1,2,1;2,4,2;1,2,1]



Task 5

- . Run your histogram program in part 2 one more time on the filtered image resulting from step 3 and obtain histogram data.

Once again I reused the exact same code I used to get the histogram for the unfiltered histogram only diffreenc is now we getting it for the filtered image the the type of intensity found in the filtered image will be stored in adress 0x40000500 and the number of pixel that have those coresponding intensity will be stored in 0x40000300

The screenshot shows the μVision4 IDE interface with several windows open:

- Registers**: Shows the current register values, including R0 through R15, CPSR, and SPSR.
- Disassembly**: Displays the assembly code for the `mini3.s` file. The code implements a histogram for a filtered image, using registers R0-R4, R6-R14, and R15 (PC) for various operations like LDRB, ADDEQ, SUBS, and BNE instructions.
- Memory**: Shows the memory dump starting at address 0x40000500, where the first few bytes are 00 3F 5F 9F BF FF 00 00 00 00 00 00 00 00 00.
- Command**: Displays the command-line interface with the command `BS \mini3\mini3 s1120`.
- System Bar**: Shows the taskbar with various application icons and system status information.

Figure 5.1 code for getting filtered histogram

The screenshot shows the Memory window titled "Memory 4" with the address 0x40000500 selected. The window displays the following byte dump:

```

Address: 0x40000500
0x40000500: 00 3F 5F 9F BF FF 00 00 00 00 00 00 00 00 00
0x40000510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Below the dump, there are tabs for Call Stack + Locals, Memory 1, Memory 3, Memory 2, and Memory 4, with Memory 4 currently selected. The status bar at the bottom shows simulation time (t1: 83.19091683 sec), clock (L:150 C:12), and memory controls (CAP NUM SCRL OVR R/W).

Figure 5.2 : 'Pixel intensity found in filterd image that bein g 0 , 63 (3F in hexa) , 95 (5F in hexa) , 159 (9F in hexa) , 191(BF in hexal) ,and 255 (FF in hexa)

Memory 4

Address: 0x40000300

```

0x40000300: 3E 30 12 12 30 3E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

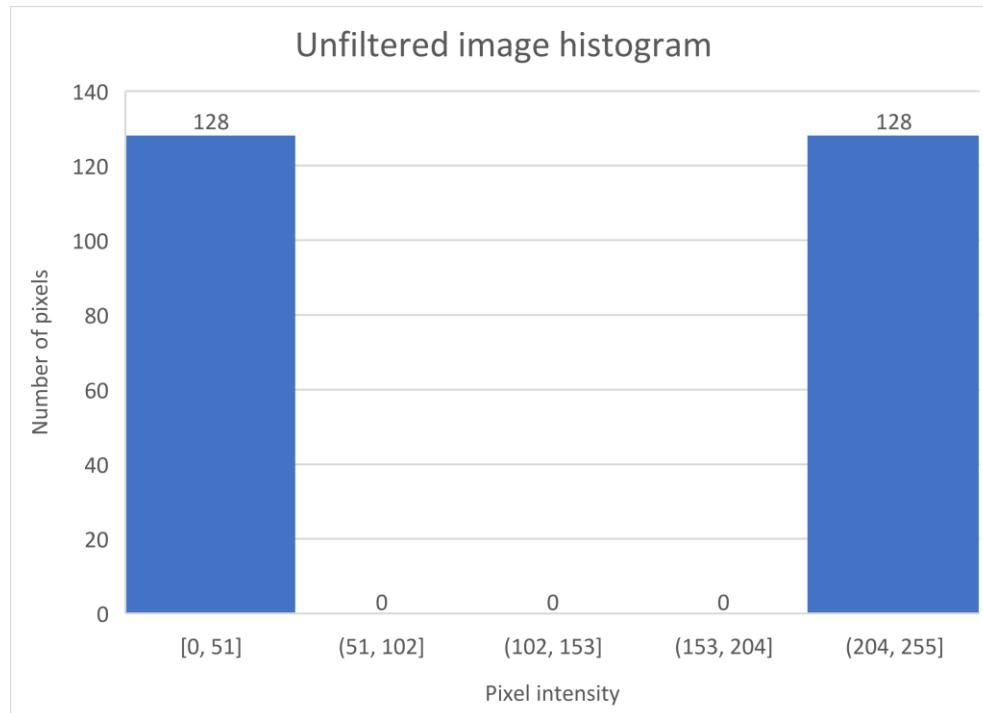
Call Stack + Locals | Memory 1 | Memory 3 | Memory 2 | **Memory 4**

Figure 5.3 number pixel found with the found intensities 62 pixel have intensity 0, 48 pixel have intensity 63 , 18 pixel have intensity 95, 18 pixel have intensity 151, 48 pixel have intensity 191 , and 62 pixel have intensity 255

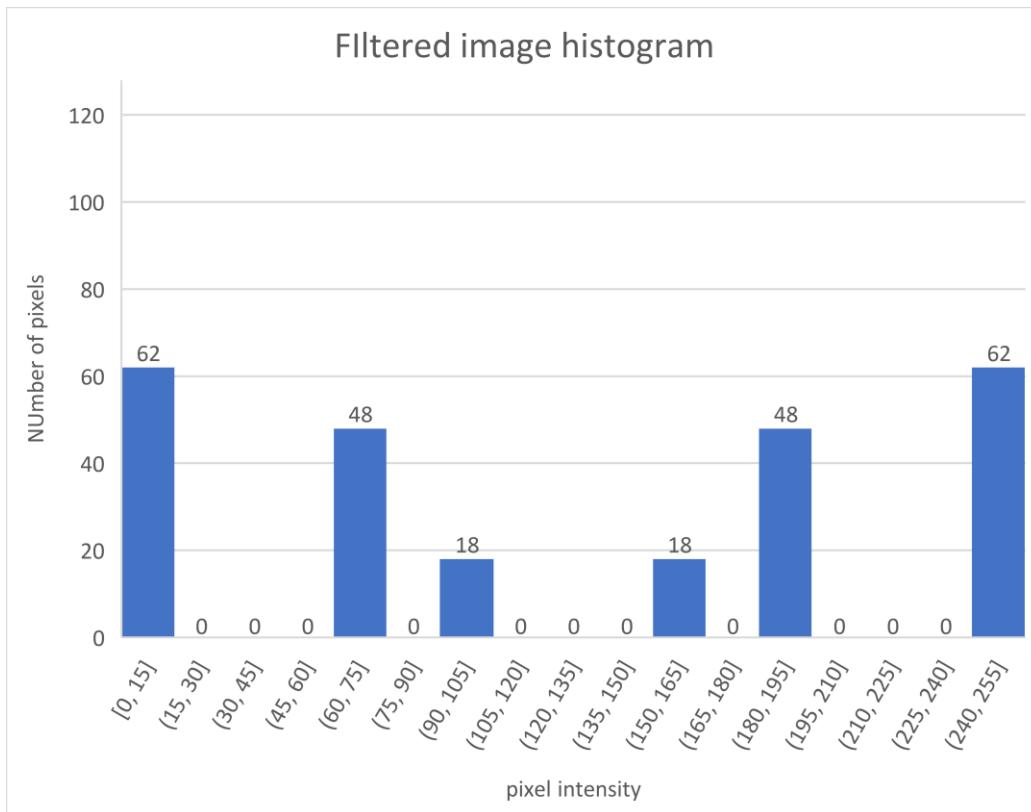
Task 6

Use Excel and data obtained in parts 2 and 5 to compare image histograms before and after applying the filter.

Unfiltered image histogram



Filtered image histogram



TASK 7

Conclusion

From doing this project I have significantly sharpened my programing skills. I was able to successfully complete all task that were asked in this project. I overall feel really accomplished for the handwork I put in to this program. The part I feel most proud of is the filter part which was defietly the most rigorous part of the project

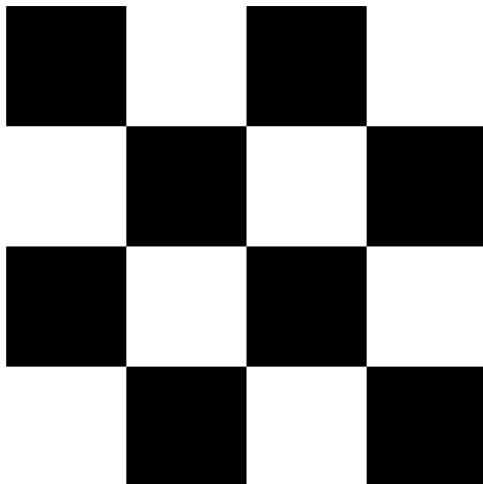
Task 8

.Use simple Matlab instructions to convert image data to images before and after filtering

Ryan mark

Tas 8 for mini project 3

```
IMAGE= [0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255; ...
255,255,255,255,0,0,0,0,0,255,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
0,0,0,0,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,0,255,255,255,0,0,0,0,255,255,255,255; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0; ...
255,255,255,255,0,0,0,0,0,255,255,255 ,0,0,0,0];
imn =uint8(IMAGE);
imshow(imn,'InitialMagnification',3000)
```

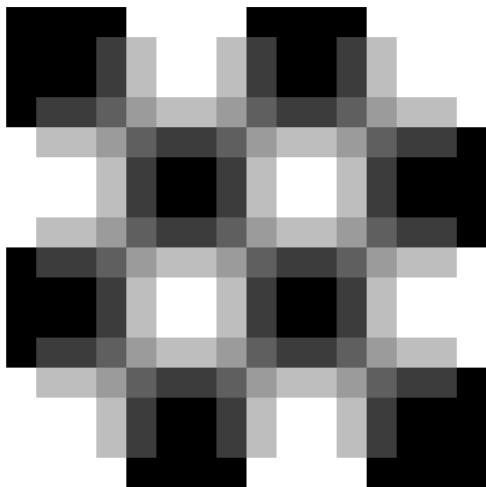


```
Filtered = [0,0,0,0,255,255,255,255,0,0,0,0,255,255,255,255; ...
0,0,0,63,191,255,255,191,63,0,0,63,191,255,255,255; ...
0,0,0,63,191,255,255,191,63,0,0,63,191,255,255,255; ...
0,63,63,95,159,191,191,159,95,63,63,95,159,191,191,255; ...
255,191,191,159,95,63,63,95,159,191,191,159,95,63,63,0; ...
255,255,255,191,63,0,0,63,191,255,255,191,63,0,0,0; ...
255,255,255,191,63,0,0,63,191,255,255,191,63,0,0,0; ...
255,191,191,159,95,63,63,95,159,191,191,159,95,63,63,0; ...
0,63,63,95,159,191,191,159,95,63,95,159,191,191,255; ...]
```

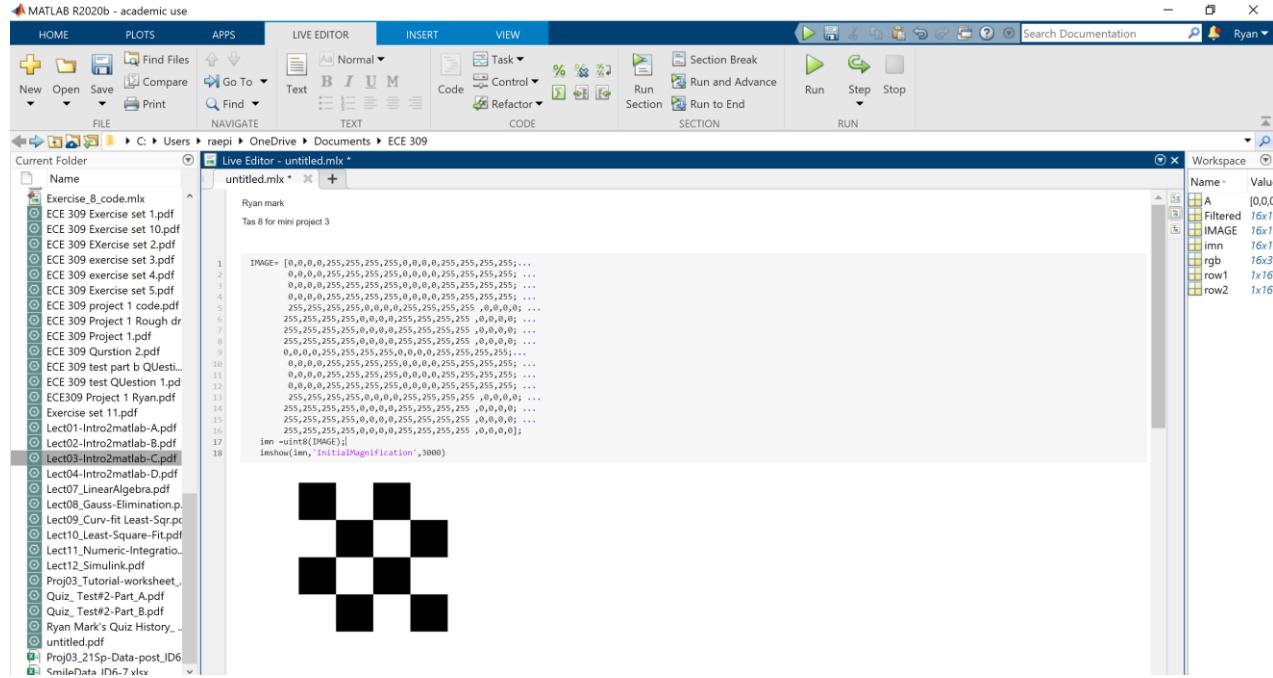
```
0,0,0,63,191,255,255,191,63,0,0,63,191,255,255,255; ...
0,0,0,63,191,255,255,191,63,0,0,63,191,255,255,255; ...
0,63,63,95,159,191,191,159,95,63,63,95,159,191,191,255; ...
255,191,191,159,95,63,63,95,159,191,191,159,95,63,63,0; ...
255,255,255,191,63,0,0,63,191,255,255,191,63,0,0,0; ...
255,255,255,191,63,0,0,63,191,255,255,191,63,0,0,0; ...
255,255,255,255,0,0,0,255,255,255,255 ,0,0,0,0];

imn =uint8(Filtered);

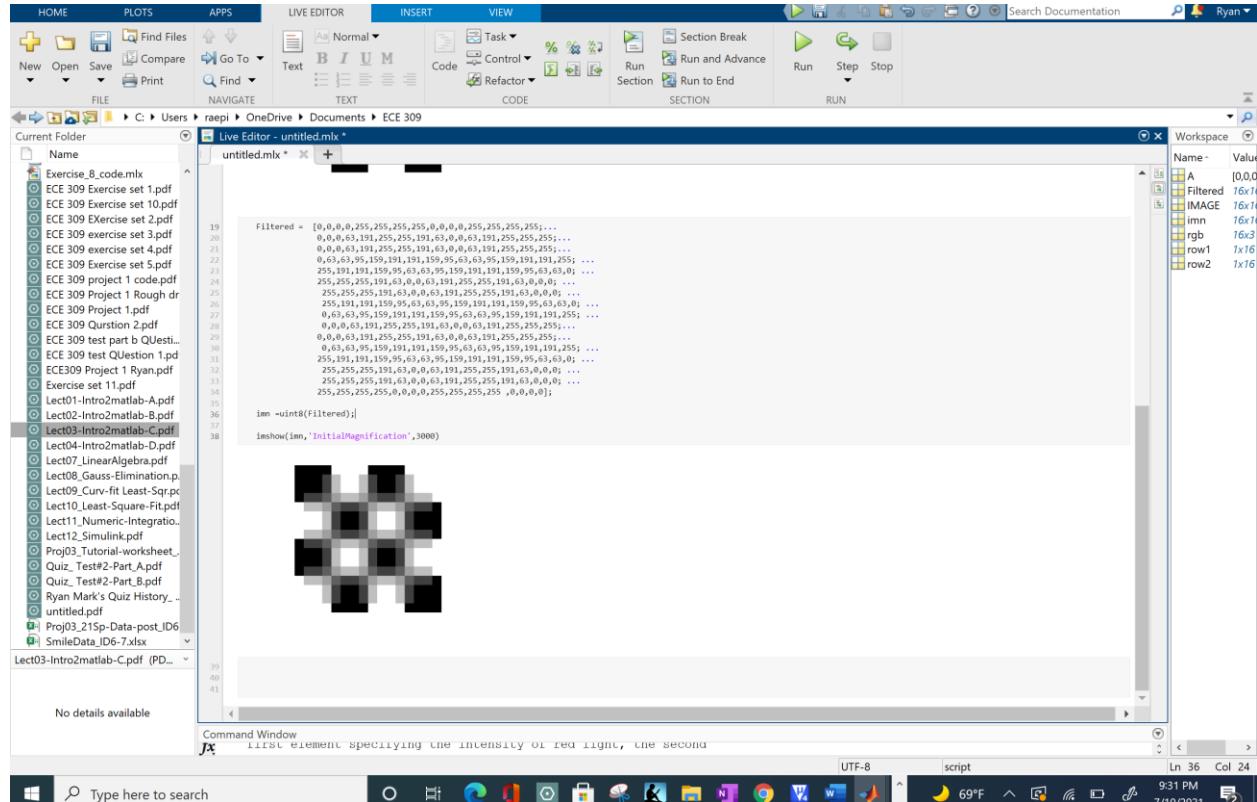
imshow(imn, 'InitialMagnification',3000)
```



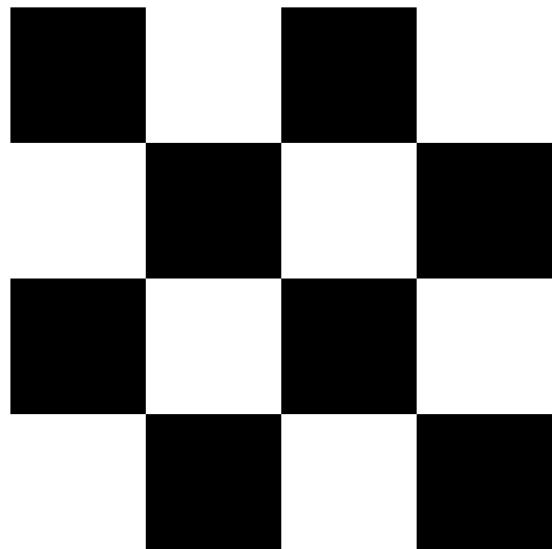
Unfiltered Image



Filtered Image



BEFORE



AFTER

