**Department of Electronic & Telecommunication Engineering**

**University of Moratuwa**

**EN 3030 – Circuits and Systems Design**



**Interim Report**

Instruction Set Architecture & Software Implementation

**Group Members**

|  |  |
| --- | --- |
| ***Name*** | ***Index Number*** |
| *Rathnayaka R.G.H.V.* | *180529E* |
| *Udara A.W.T.* | *180650P* |
| *Thilakarathna G.D.O.L* | *180642T* |
| *Sewwandi B.L.P.N.* | *180589K* |

December 15, 2021

*This Report is submitted as a partial fulfillment of the requirements for the module EN 3030 – Circuits and Systems Design*

*Department of Electronic and Telecommunication Engineering,*

University of Moratuwa.

**Introduction to the Task**

The task is focusing on the design of a processor which suits to down sample a grayscale image of size n x n pixels to ( pixels. We selected to down sample a grayscale image of 256 x 256 pixels to 128 x 128 pixels. For continuing the task, we designed a Gaussian low pass filter in the aim of reducing any information loss and minimizing the changes happening because of aliasing due to high frequency components.

**Algorithms Associated with the Task**

Algorithm for the processing part of the task is developed based on two parts as,

1. Filtering Algorithm
2. Down Sampling Algorithm

All the testing regarding these two algorithms was done using **Python 3.9**.

1. Filtering Algorithm

As the very first part of the task, we had to remove the high pass frequency components in the image. For that, we used a Gaussian low pass filter which can remove the high pass frequency components. The Gaussian filter that was used for filtering in this implementation is given below under *Figure 01* with the weight distribution of the kernel.

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 1 |
| 3 | 16 | 3 |
| 1 | 3 | 1 |

Figure 0 – 3x3 Gaussian Kernel used in Low pass Filtering

The selected image of N x N pixels was convoluted with this 3 x 3 Gaussian kernel which outputs an image where low pass component are there. In this convolution process, overlapping the center pixel of the kernel with an image pixel provides a weighted summation value. This result is divided by the total weight of the kernel (here it is 32.) to normalize. Then, this average value is stored at the top left corner pixel location in the RAM. After the initial convolution operation, the kernel is moving forward and the average value of the each set of convolutions is stored at the top left corner pixel. After finishing the forward move kernel move downwards and again the procedure is continuing until all the set of pixels are covered. Filtered image is completely stored in the RAM in this sequential order. As we neglected the effect surrounding marginal pixels’ effect. So, no padding was done for the 3 x 3 kernel.

Python implementation done with **Jupyter Notebook 6.0** to filter the image to avoid high frequency components is given below.

1. Down Sampling Algorithm

Since, it is needed to down sample the image from both height and width with a down sampling factor of 2, a one value is taken per four pixels from the image obtained through convolution with the 3 x 3 kernel. Then, that value is kept stored in the memory for getting the output. Values taken from a 6 x 6 filtered image is shown in the *Figure 02* below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Figure 02 – Graphical Illustration of Down Sampling method

Every other pixel from the filtered image is used in constructing the output image. No interpolation is used as the image is already down sampled. The down sampling algorithm done with the use of python through **Jupyter Notebook 6.0** is given below.

**Hardware Requirements**

1. Registers

Implementation of the processor design was done with the use of 5 special purpose registers.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Size | Used to store | |
| When Filtering | When Sampling |
| Z | 1 bit | Control the flow | Control the flow |
| AC | 16 bits | Inputs to ALU | Inputs to ALU |
| R | 16 bits | Calculated data |  |
| R0 | 16 bits | Addresses to read memory |  |
| R1 | 16 bits | Limit of no of loops | Addresses to read memory |
| R2 | 16 bits |  | Addresses to write memory |
| R3 | 16 bits |  | No of rows in down sampled image |
| R4 | 16 bits | Addresses to write memory | No of columns in down sampled image |

Table 0 – Information on Registers

1. Memory

Two types of memory are associated with this design as,

1. Instruction Memory – keeping the instruction in the program.
2. Data Memory – Storing data on intensity values of the pixels of input and output image. Here, size of a memory location is 8 bits while the address length comprises of 16 bits.

Instructions are in frame size of 8 bits (1 byte). 32 types of instructions are used in ISA. The minimum size expected from the data memory can be evaluated as,

* Total number of pixels in the input image--------------------------256 x 256
* Intensity value width per pixel----------------------------------------- 8 bits
* Memory size needed to store input image----------------------256 x 256 x 8 bits

**Instruction Set Architecture (ISA)**

|  |  |  |
| --- | --- | --- |
| INSTRUCTION | INSTRUCTION CODE | INSTRUCTION OPERATION |
| NOP | 0000 0000 | No operation |
| CLAC | 0000 0001 | AC←0  Z←1 |
| LDAC 𝒯 | 0000 0010 𝒯 | AC←M[𝒯] |
| STAC 𝒯 | 0000 0011 𝒯 | M[𝒯]←AC |
| ADDM α | 0000 0100 α | AC←AC+α |
| SUBM α | 0000 0101 α | AC← AC-α |
| DIV 32 | 0000 0110 | AC←AC/32 |
| ADD | 0000 0111 | AC←AC+R |
| MUL | 0000 1000 | AC←AC\*R |
| MULM α | 0000 1001 α | AC ← AC\*α |
| INCR0 | 0000 1010 | R0←R0+1 |
| INCR1 | 0000 1011 | R1←R1+1 |
| INCR2 | 0000 1100 | R2←R2+1 |
| INCR4 | 0000 1101 | R4←R4+1 |
| DECR0 | 0000 1110 | R0←R0-2 |
| DECR | 0000 1111 | R←R-1 if (R-1 =0) Z←0 else Z←1 |
| DECR1 | 0001 0000 | R1←R1-1 if (R1-1 =0) Z←0 else Z←1 |
| DECR3 | 0001 0001 | R3←R3-1 if (R3-1 =0) Z←0 else Z←1 |
| DECR4 | 0001 0010 | R4←R4-1 if (R4-1 =0) Z←0 else Z←1 |
| MVACR | 0001 0011 | R←AC |
| MVACR0 | 0001 0100 | R0←AC |
| MVACR1 | 0001 0101 | R1←AC |
| MVACR2 | 0001 0111 | R2←AC |
| MVACR3 | 0001 1000 | R3←AC |
| MVACR4 | 0001 1001 | R4←AC |
| MVRAC | 0001 1010 | AC←R |
| MVR1AC | 0001 1011 | AC←R1 |
| MVR2AC | 0001 1100 | AC←R2 |
| MVR3AC | 0001 1101 | AC←R3 |
| MVR4AC | 0001 1110 | AC←R4 |
| JPNZ 𝒯 | 0001 1111 𝒯 | IF (Z==0) GOTO 𝒯 |
| JPPZ 𝒯 | 0010 0000 𝒯 | IF (Z==1) GOTO 𝒯 |

Table 0 – Instruction Set

**Assembly Code**

Implementation of the aforementioned Python code was done with the use of some assembly level codes. The code segment used in implementation using multiple instructions from ISA is given below.

*[ This code is to down sample a grayscale image of size 256 x 256 pixel to an image of 128 x 128 pixels]*

1. *CLAC ;  AC←0, Z←1*
2. *ADDM 256 ; AC←AC+256*
3. *MVACR ; R←AC*
4. *DECR ; R← R-1*
5. *DECR ; R ← R-1*
6. *MUL ; AC←AC\*R*
7. *MVACR1 ; R1←AC // limit for filtering*
8. *DECR1 ; R1← R1-1*
9. *DECR1 ; R1← R1-1*
10. *CLAC; AC←0, Z←1*
11. *MVACR0 ; R0←AC // read address*
12. *MVACR3 ; R3 ← AC*
13. *MVACR4 ; R4 ← AC // middle pixel address*
14. *INCR4 ; R4 ← R4+1*
15. *MVR4AC ; AC← R4*
16. *ADDM 256 ; AC← AC+256 // initial pixel address*
17. *MVACR4 ; R4← AC*
18. *LOOP:CLAC ; AC ←0, Z←1*
19. *MVACR ; R←AC*
20. *MVR4AC ; AC← R4*
21. *MVACR0 ; R0← AC*
22. *LDAC R0 ; AC ← M[R0]*
23. *MULM 16 ; AC← AC\*16*
24. *MVACR ; R← AC*
25. *INCR0 ; R0← R0+1*
26. *LDAC R0 ; AC← M[R0]*
27. *MULM 3 ; AC← AC\*3*
28. *ADD ; AC← AC+R*
29. *MVACR ; R← AC*
30. *DECR0 ; R0← R0-2*
31. *LDAC R0; AC← M[R0]*
32. *MULM 3 ; AC← AC\*3*
33. *ADD ; AC← AC+R*
34. *MVACR ; R← AC*
35. *MVR4AC ; AC← R4*
36. *ADDM 256 ; AC← AC+256*
37. *MVACR0 ; R0← AC*
38. *LDAC R0 ;AC← M[R0]*
39. *MULM 3 ;AC← AC\*3*
40. *ADD ; AC← AC+R*
41. *MVACR ; R← AC*
42. *INCR0; R0← R0+1*
43. *LDAC R0; AC← M[R0]*
44. *ADD ; AC← AC+R*
45. *MVACR ; R ← AC*
46. *DECR0 ; R0← R0-2*
47. *LDAC R0 ; AC← M[R0]*
48. *ADD ; AC← AC+R*
49. *MVACR ; R← AC*
50. *MVR4AC ; AC← R4*
51. *SUBM 256 ; AC← AC-256*
52. *MVACR0 ; R0← AC*
53. *LDAC R0 ;AC← M[R0]*
54. *MULM 3 ;AC← AC\*3*
55. *ADD ; AC← AC+R*
56. *MVACR ; R← AC*
57. *INCRO; R0← R0+1*
58. *LDAC R0; AC← M[R0]*
59. *ADD ; AC← AC+R*
60. *MVACR ; R ← AC*
61. *DECR0 ; R0← R0-2*
62. *LDAC R0 ; AC← M[R0]*
63. *ADD ; AC← AC+R*
64. *DIV 32; AC← AC/32*
65. *STAC R4 ;M[R4]  ← AC*
66. *INCR4 ; R4 ← R4+1*
67. *DECR1 ;R1←R1-1,  if (R1-1 =0) Z←0 else Z←1*
68. *JPPZ LOOP(18)*

*Filtering of image is over. After filtering down sampling of image by 2 is as below as following assembly code.*

1. *CLAC ; AC←0, Z←1*
2. *ADDM 128; AC←AC+128*
3. *MVACR3 ;R3 ← AC // No of rows  in down sampled image*
4. *MCAVR4 ; R4←AC // No of columns in down sampled image*
5. *CLAC ; AC←0, Z←1*
6. *MVACR1 ; R1←AC // READ ADDRESS*
7. *MVACR2 ; R2 ← AC // WRITE ADDRESS*
8. *LOOP:LDAC R1 ;AC←M[R1]*
9. *STAC R2 ; M[R2]←AC*
10. *INCR2 ; R2←R2+1*
11. *INCR1 ; R1←R1+1*
12. *INCR1 ; R1←R1+1*
13. *DECR4 ; R4←R4-1 if (R4-1 =0) Z←0 else Z←1*
14. *JPPZ 76*
15. *MVR1AC ; AC←R1*
16. *ADDM 256 ; AC←AC+256*
17. *MVACR1 ; AC←R1*
18. *CLAC; AC←0*
19. *ADDM 128 ; AC←AC+128*
20. *MVACR4; R4←AC*
21. *DECR3 ; R3←R3-1 if(R3-1=0) Z←0 else Z←1*
22. *JPPZ 76*