// properties of the original image

int image\_width = 4; // width of the image

int image\_height = 4; // height of the image

// downsampled image

int downsampledimage[4]; // variable to store downsampled image

// properties of the downsampled image

int dsimage\_width = 2; // width of the downsampled image

int dsimage\_height = 2; // height of the downsampled image

int main(){

//Horizontal Convolution

printf("Horizontal Convolution\n");

int height\_count = image\_height;

int x = 0;

while (height\_count > 0){

int y = 0;

int a = 0; //zero padding(left)

int b = 2 \* image[x\*image\_width + y];

int width\_count = image\_width - 1;

while (width\_count > 0){

int c = image[x\*image\_width + y + 1];

int new\_pixel = (a + b + c)/4;

image[x\*image\_width + y] = new\_pixel;

printf("%i, ", new\_pixel);

//sliding window

a = b/2;

b = c\*2;

y += 1; //moving to next pixel

width\_count -= 1;

}

int c = 0; //zero padding(right)

int new\_pixel = (a + b + c)/4;

image[x\*image\_width + y] = new\_pixel;

printf("%i, ", new\_pixel);

height\_count -= 1;

x += 1; // moving to next row

printf("\n");

}

//Vertical Convolution

printf("\nVertical Convolution\n");

int width\_count = image\_width;

int y = 0;

while (width\_count > 0){

int x = 0;

int a = 0; //zero padding(top)

int b = 2\*image[x\*image\_width + y];

int height\_count = image\_height - 1;

while (height\_count > 0){

int c = image[x\*image\_width + image\_width + y];

int new\_pixel = (a + b + c)/4;

image[x\*image\_width + y] = new\_pixel;

printf("%i, ", new\_pixel);

//sliding window

a = b/2;

b = c\*2;

x += 1; //moving to next pixel

height\_count -= 1;

}

int c = 0; //zero padding(bottom)

int new\_pixel = (a + b + c)/4;

image[x\*image\_width + y] = new\_pixel;

printf("%i, ", new\_pixel);

width\_count -= 1;

y += 1; // moving to next column

printf("\n");

}

// downsampling

printf("\nDownsampling\n");

height\_count = dsimage\_height;

x = 0;

while (height\_count > 0){

int y = 0;

int width\_count = dsimage\_width;

while (width\_count > 0){

int pixel\_value = image[2\*y\*image\_width + 2\*x];

downsampledimage[x\* dsimage\_width + y] = pixel\_value;

printf("%i, ", pixel\_value);

y += 1; // moving to next pixel

width\_count -= 1;

}

height\_count -= 1;

x += 1; // moving to next row

printf("\n");

}

base\_address(f[0][0]) - x1

downsampled\_image(g[0][0]) - x4

image\_height - x2

image\_width - x3

height\_count - x5

pixel\_address - x6

a = x7

b = x8

c = x9

Downsampled\_height = x10

Downsampled\_width = x11

width\_count = x12

column\_base\_address = x13

next\_pixel\_address = x14

| INSTRUCTION NUM | HIGH LEVEL CODE | ASSEMBLY CODE | U INSTRUCTION |
| --- | --- | --- | --- |
| 1 |  | CLAC | AC ← 0 |
| 2 | int image[]; // original image | LD X1,R1 | MAR <~ X1  READ  DR ← M(X1)  AC ← DR  R1 ← AC |
| 3 | int image\_width = 4; | LD X2,R2 | MAR <~ X2  READ  DR ← M(X2)  AC ← DR  R2 ← AC |
| 4 | int image\_height = 4; | LD X3,R3 | MAR <~ X3  READ  DR ← M(X3)  AC ← DR  R3 ← AC |
| 5 | int height\_count = image\_height; | SW X5,R3 | AC<~R3  AR<~X5  DR <~ AC  WRITE |
| 6 |  | CLAC | AC<~ 0 |
| 7 | int X =0 | MOV AC,R4 | R4 <~AC |
| 8 | Int y = 0 | MOV AC,R5 | R5 <~AC |
| 9 | Int a = 0 | MOV AC,R6 | R6 <~AC |
| 10 | int b = 2 \* image[x\*image\_width + y];// value b is stored in “R7” | MOV R2AC | AC <~R2 |
| 11 | MUL R4 | AC <~ AC \*R4 |
| 12 | ADD R5 | AC <~AC +R5 |
| 13 | ADD R1 | AC <~ AC+R1 |
| 14 | LDAC | AR ← AC  READ  DR ← M(AC)  AC ← DR |
| 15 | LSHIFT | AC <~ AC << 1 |
| 16 | MOV AC,R7 | R7<~AC |
| 17 | int width\_count = image\_width - 1; | MOV R2,AC | AC<~R2 |
| 18 | DECREMENT AC | AC<~AC -1 |
| 19 | SW X12,AC | AR<~X12  DR <~ AC  WRITE |
| 20 | int c = image[x\*image\_width + y + 1]; // c is stored in R8 | MOV R2,AC | AC <~R2 |
| 21 | MUL R4 | AC <~ AC \*R4 |
| 22 | ADD R5 | AC <~AC +R5 |
| 23 | INCREMENT AC | AC<~AC+1 |
| 24 | ADD R1 | AC <~ AC+R1 |
| 25 | LDAC | AR ← AC  READ  DR ← M(AC)  AC ← DR |
| 26 | MOV AC,R8 | R8<~AC |
| 27 | int new\_pixel = (a + b + c)/4;  // new pixel value will be stored in R9 | ADD R7 | AC <~ AC+R7 |
| 28 | ADD R6 | AC <~ AC+R6 |
| 29 | RSHIFT | AC <~ AC >> 1 |
| 30 | RSHIFT | AC <~ AC >> 1 |
| 31 | MOV AC,R9 | R9 <~AC |
| 32 | image[x\*image\_width + y] = new\_pixel; | MOV R2,AC | AC <~R2 |
| 33 | MUL R4 | AC <~ AC \*R4 |
| 34 | ADD R5 | AC <~ AC+R5 |
| 35 | ADD R1 | AC <~ AC+R1 |
| 36 | MOV AC,MAR | MAR<~AC |
| 37 | MOV R9,AC | AC <~ R9 |
| 38 | MOV AC,MDR | MDR <~AC  WRITE |
| 39 | a = b/2; | MOV R7,AC | AC<~R7 |
| 40 | RSHIFT | AC <~ AC >> 1 |
| 41 | MOV AC,R6 | R6<~AC |
| 42 | b = c\*2; | MOV R8,AC | AC<~R8 |
| 43 | LSHIFT | AC <~ AC << 1 |
| 44 | MOV AC,R7 | R7<~AC |
| 45 | y += 1; | MOV R5,AC | AC<~R5 |
| 46 | INCREMENT AC | AC<~AC+1 |
| 47 | MOV AC,R5 | R5<~AC |
| 48 | width\_count -= 1; | LD X12,R9 | MAR <~ X12  READ  DR ← M(X12)  AC ← DR  R9 ← AC |
| 49 | MOV R9,AC | AC<~R9 |
| 50 | DECREMENT AC | AC<~AC-1 |
| 51 | STAC | DR <~ AC  WRITE |
| 52 | while (width\_count > 0): repeat from 20 if the flag is not zero. | JMPNZ 20 | AC <~ IM(t)  PC <~ AC  PC <~PC+1 |
| 53 | int c = 0; //zero padding(right)  int new\_pixel = (a + b + c)/4; | CLAC | AC<~0 |
| 54 | ADD R7 | AC <~ AC+R7 |
| 55 | ADD R6 | AC <~ AC+R6 |
| 56 | RSHIFT | AC <~ AC >> 1 |
| 57 | RSHIFT | AC <~ AC >> 1 |
| 58 | MOV AC,R9 | R9 <~AC |
| 59 | image[x\*image\_width + y] = new\_pixel; | MOV R2,AC | AC <~R2 |
| 60 | MUL R4 | AC <~ AC \*R4 |
| 61 | ADD R5 | AC <~ AC+R5 |
| 62 | ADD R1 | AC <~ AC+R1 |
| 63 | MOV AC,MAR | MAR<~AC |
| 64 | MOV R9,AC | AC <~ R9 |
| 65 | MOV AC,MDR | MDR <~AC  WRITE |
| 66 | height\_count -= 1; | LD X5,R9 | MAR <~ X5  READ  DR ← M(X5)  AC ← DR  R9 ← AC |
| 67 | MOV R9,AC | AC<~R9 |
| 68 | DECREMENT AC | AC<~AC-1 |
| 69 | STAC | DR <~ AC  WRITE |
| 70 | x += 1; // moving to next row | MOV R4,AC | AC<~R4 |
| 71 | INCREMENT AC | AC<~AC+1 |
| 72 | MOV AC,R4 | R4<~AC |
| 73 | while (height\_count > 0): repeat from step 8 | JMPNZ 8 | AC <~ IM(t)  PC <~ AC  PC <~PC+1 |
| 74 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |