```
// 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			
