```c
// 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){
```

```c
    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");
}
```

image_height - x2 ——————————————————————DATA
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
HORIZONTAL CONVOLUTION			
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	x += 1; // moving to next row	MOV R4,AC	AC<~R4
67		INCREMENT AC	AC<~AC+1
68		MOV AC,R4	R4<~AC
69	height_count -= 1;	LD X5,R9	MAR <~ X5 READ DR ← M(X5) AC ← DR R9 ← AC
70		MOV R9,AC	AC<~R9
71		DECREMENT AC	AC<~AC-1
72		STAC	DR <~ AC WRITE

73	while (height_count > 0): repeat from step 8	JMPNZ 8	AC <~ IM(t) PC <~ AC PC <~PC+1
VERTICAL CONVOLUTION			
74	int width_count = image_width;	SW X12,R2	AC<~R2 AR<~X12 DR <~ AC WRITE
75	int y = 0;	CLAC	AC<~0
76		MOV AC,R5	R5<~AC
77	int x = 0;	MOV AC,R4	R4<~AC
78	int a = 0; //zero padding(top)	MOV AC,R6	R6 <~AC
79	int b = 2*image[x*image_width + y];	MOV R2AC	AC <~R2
80		MUL R4	AC <~ AC *R4
81		ADD R5	AC <~AC +R5
82		ADD R1	AC <~ AC+R1
83		LDAC	AR ← AC READ DR ← M(AC) AC ← DR
84		LSHIFT	AC <~ AC << 1
85		MOV AC,R7	R7<~AC
86	int height_count = image_height - 1;	MOV R3,AC	AC<~R3
87		DECREMENT AC	AC<~AC -1
88		SW X5,AC	AR<~X5 DR <~ AC WRITE
89	int c = image[x*image_width + image_width + y];	MOV R2,AC	AC <~R2
90		MUL R4	AC <~ AC *R4
91		ADD R5	AC <~AC +R5

92		ADD R2	AC <~AC+R2
93		ADD R1	AC <~ AC+R1
94		LDAC	AR ← AC READ DR ← M(AC) AC ← DR
95		MOV AC,R8	R8<~AC
96	int new_pixel = (a + b + c)/4; // <mark>new pixel value will be stored in R9</mark>	ADD R7	AC <~ AC+R7
97		ADD R6	AC <~ AC+R6
98		RSHIFT	AC <~ AC >> 1
99		RSHIFT	AC <~ AC >> 1
100		MOV AC,R9	R9 <~AC
101	a = b/2;	MOV R7,AC	AC<~R7
102		RSHIFT	AC <~ AC >> 1
103		MOV AC,R6	R6<~AC
104	b = c*2;	MOV R8,AC	AC<~R8
105		LSHIFT	AC <~ AC << 1
106		MOV AC,R7	R7<~AC
107	X+=1	MOV R4,AC	AC<~R4
108		INCREMENT AC	AC<~AC+1
109		MOV AC,R4	R4<~AC
110	height_count -= 1;	LD X5,R9	MAR <~ X5 READ DR ← M(X5) AC ← DR R9 ← AC
111		MOV R9,AC	AC<~R9
112		DECREMENT AC	AC<~AC-1
113		STAC	DR <~ AC WRITE

114	while (height_count > 0): repeat from step 89	JMPNZ 89	AC <~ IM(t) PC <~ AC PC <~PC+1

115	int c = 0; //zero padding(right)
	int new_pixel = (a + b + c)/4;
		CLAC	AC<~0

116		ADD R7	AC <~ AC+R7
117		ADD R6	AC <~ AC+R6
118		RSHIFT	AC <~ AC >> 1
119		RSHIFT	AC <~ AC >> 1
120		MOV AC,R9	R9 <~AC
121	image[x*image_width + y] = new_pixel;	MOV R2,AC	AC <~R2
122		MUL R4	AC <~ AC *R4
123		ADD R5	AC <~ AC+R5
124		ADD R1	AC <~ AC+R1
125		MOV AC,MAR	MAR<~AC
126		MOV R9,AC	AC <~ R9
127		MOV AC,MDR	MDR <~AC WRITE
128	Y += 1; // moving to next row	MOV R5,AC	AC<~R5
129		INCREMENT AC	AC<~AC+1
130		MOV AC,R5	R5<~AC
131	width_count -= 1;	LD X12,R9	MAR <~ X12 READ DR ← M(X12) AC ← DR R9 ← AC
132		MOV R9,AC	AC<~R9
133		DECREMENT AC	AC<~AC-1
134		STAC	DR <~ AC WRITE

135	while (width_count > 0): repeat from 77 if the flag is not zero.	JMPNZ 77	AC <~ IM(t) PC <~ AC PC <~PC+1
DOWNSAMPLING			
136	int downsampledimage[4]; // base address to store the downsampled image	LD X4,R6	MAR <~ X4 READ DR ← M(X4) AC ← DR R6 ← AC
137	height_count = image_height/2; // downsampled image_height;	MOV R3,AC	AC<~R3
138		RSHIFT	AC<~ AC>>1
139		SW X5,AC	AR<~X5 DR <~ AC WRITE
140	x = 0;	CLAC	AC<~0
141		MOV AC,R4	R4<~AC
142	int y = 0;	MOV AC,R5	R5<~AC
143	int width_count = image_width/2; // dsimage_width;	MOV R2,AC	AC<~R2
144		RSHIFT	AC<~ AC>>1
145		SW X12,AC	AR<~X12 DR <~ AC WRITE
146	int pixel_value = image[2*y*image_width + 2*x];	MOV R2,AC	AC<~R2
147		MUL R5	AC <~ AC *R5
148		ADD R4	AC <~ AC +R4
149		LSHIFT	AC <~ AC << 1
150		ADD R1	AC <~ AC +R1
151		LDAC	AR ← AC READ DR ← M(AC) AC ← DR
152		MOV AC,R9	R9<~AC

153	downsampledimage[x* (image_width/2) + y] = pixel_value;	MOV R2,AC	AC<~R2
154		RSHIFT	AC<~ AC>>1
155		MUL R4	AC <~ AC *R4
156		ADD R5	AC <~ AC +R5
157		ADD R6	AC <~ AC +R6
158		MOV AC,MAR	MAR<~AC
159		MOV R9,AC	AC <~ R9
160		MOV AC,MDR	MDR <~AC WRITE
161	y += 1; // moving to next pixel	MOV R5,AC	AC<~R5
162		INCREMENT AC	AC<~AC+1
163		MOV AC,R5	R5<~AC
164	width_count -= 1;	LD X12,R9	MAR <~ X12 READ DR ← M(X12) AC ← DR R9 ← AC
165		MOV R9,AC	AC<~R9
166		DECREMENT AC	AC<~AC-1
167		STAC	DR <~ AC WRITE
168	while (width_count > 0): repeat from step 146	JMPNZ 146	AC <~ IM(t) PC <~ AC PC <~PC+1
169	x += 1; // moving to next row	MOV R4,AC	AC<~R4
170		INCREMENT AC	AC<~AC+1
171		MOV AC,R4	R4<~AC
172	height_count -= 1;	LD X5,R9	MAR <~ X5 READ DR ← M(X5) AC ← DR R9 ← AC

173		MOV R9,AC	AC<~R9
174		DECREMENT AC	AC<~AC-1
175		STAC	DR <~ AC WRITE
176	while (width_count > 0): repeat from step 142	JMPNZ 142	AC <~ IM(t) PC <~ AC PC <~PC+1