Algorithm

Define varibles

```
// 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
```

Horizontal convolution

```
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

```
//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 18'd20 ----------ADDRESS

downsampled_image(g[0][0]) - x4 18d'65536+8;--ADDRESS
image_height - x2 18'd1; ----------DATA
image_width - x3 18'd2;
height_count - x5 18'd3;
a = x7 18d'4;
b = x8 18d'5;
c = x9 18d'6;
width_count = x12 18d'7;
pixel_address - x6
Downsampled_height = x10
Downsampled_width = x11
column_base_address = x13
next_pixel_address = x14

MICROINSTRUCTIONS

INSTRUCTION NUM	HIGH LEVEL CODE	ASSEMBLY CODE	U INSTRUCTION
HORIZONTAL CONVOLUTION			
1		CLAC	AC ← 0
2	int image[]; // original image	LD X1,R1	MAR ← MBRU ;READ IDLE DR ← M(X1) AC ← DR R1 ← AC
3	int image_width = 4;	LDI R2	AC← MBRU R2 ← AC
4	int image_height = 4;	LDI R3	AC ← MBRU R3 ← AC
5	int height_count = image_height;	SW X5,R3	AC←R3 AR← MBRU

			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 R4AC	AC ←R4
11		LSHIFT8	AC ← AC<<7
12		ADD R5	AC ←AC +R5
13		ADD R1	AC ← AC+R1
14		LDAC	AR ← AC; READ IDLE DR ← M(AC) AC ← DR
15		LSHIFT1	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← MBRU DR ←AC WRITE
20	int c = image[x*image_width + y + 1]; // c is stored in R8	MOV R4,AC	AC ←R4
21		LSHIFT8	AC ← AC<<7
22		ADD R5	AC ←AC +R5
23		INCREMENT AC	AC←AC+1
24		ADD R1	AC ← AC+R1
25		LDAC	AR ← AC; READ IDLE 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		RSHIFT1	AC ← AC >> 1
30		RSHIFT1	AC ← AC >> 1
31		MOV AC,R9	R9 ←AC
32	image[x*image_width + y] = new_pixel;	MOV R4,AC	AC ←R4
33		LSHIFT8	AC ← AC<<7
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		RSHIFT1	AC ← AC >> 1
41		MOV AC,R6	R6←AC
42	b = c*2;	MOV R8,AC	AC←R8
43		LSHIFT1	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 ← MBRU ; READ IDLE 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 ←MBRU PC ← AC

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		RSHIFT1	AC ← AC >> 1
57		RSHIFT1	AC ← AC >> 1
58		MOV AC,R9	R9 ←AC
59	image[x*image_width + y] = new_pixel;	MOV R4,AC	AC ←R4
60		LSHIFT8	AC ← AC<<7
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 ← MBRU; READ IDLE 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 ← MBRU PC ← AC
VERTICAL CONVOLUTION			
74	int width_count = image_width;	SW X12,R2	AC←R2 AR← MBRU 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 R4,AC	AC ←R4
80		LSHIFT8	AC ← AC<<7
81		ADD R5	AC ←AC +R5
82		ADD R1	AC ← AC+R1
83		LDAC	AR ← AC;READ IDLE DR ← M(AC) AC ← DR
84		LSHIFT1	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← MBRU DR ← AC WRITE
89	int c = image[x*image_width + image_width + y];	MOV R4,AC	AC ←R4
90		LSHIFT8	AC ← AC<<7

91		ADD R5	AC ←AC +R5
92		ADD R2	AC ←AC+R2
93		ADD R1	AC ← AC+R1
94		LDAC	AR ← AC; READ IDLE DR ← M(AC) AC ← DR
95		MOV AC,R8	R8←AC
96	int new_pixel = (a + b + c)/4; // new pixel value will be stored in R9	ADD R7	AC ← AC+R7
97		ADD R6	AC ← AC+R6
98		RSHIFT1	AC ← AC >> 1
99		RSHIFT1	AC ← AC >> 1
100		MOV AC,R9	R9 ←AC
101	image[x*image_width + y] = new_pixel;	MOV R4,AC	AC ←R4
102		LSHIFT8	AC ← AC<<8
103		ADD R5	AC ← AC+R5
104		ADD R1	AC ← AC+R1
105		MOV AC,MAR	MAR←AC
106		MOV R9,AC	AC ← R9
107		MOV AC,MDR	MDR ←AC WRITE
108	a = b/2;	MOV R7,AC	AC←R7
109		RSHIFT1	AC ← AC >> 1
110		MOV AC,R6	R6←AC
111	b = c*2;	MOV R8,AC	AC←R8
112		LSHIFT1	AC ← AC << 1
113		MOV AC,R7	R7←AC

114	X+=1	MOV R4,AC	AC←R4
115		INCREMENT AC	AC←AC+1
116		MOV AC,R4	R4←AC
117	height_count -= 1;	LD X5,R9	MAR ← MBRU; READ IDLE DR ← M(X5) AC ← DR R9 ← AC
118		MOV R9,AC	AC←R9
119		DECREMENT AC	AC←AC-1
120		STAC	DR ← AC WRITE
121	while (height_count > 0): repeat from step 89	JMPNZ 89	AC ← MBRU PC ← AC

122	int c = 0; //zero padding(right)
	int new_pixel = (a + b + c)/4;
		CLAC	AC←0

123		ADD R7	AC ← AC+R7
124		ADD R6	AC ← AC+R6
125		RSHIFT1	AC ← AC >> 1
126		RSHIFT1	AC ← AC >> 1
127		MOV AC,R9	R9 ←AC
128	image[x*image_width + y] = new_pixel;	MOV R4,AC	AC ←R4
129		LSHIFT8	AC ← AC<<7
130		ADD R5	AC ← AC+R5
131		ADD R1	AC ← AC+R1
132		MOV AC,MAR	MAR←AC
133		MOV R9,AC	AC ← R9
134		MOV AC,MDR	MDR ←AC WRITE
135	Y += 1; // moving to next row	MOV R5,AC	AC←R5

136		INCREMENT AC	AC←AC+1
137		MOV AC,R5	R5←AC
138	width_count -= 1;	LD X12,R9	MAR ← MBRU ;READ IDLE DR ← M(X12) AC ← DR R9 ← AC
139		MOV R9,AC	AC←R9
140		DECREMENT AC	AC←AC-1
141		STAC	DR ← AC WRITE
142	while (width_count > 0): repeat from 77 if the flag is not zero.	JMPNZ 77	AC ← MBRU PC ← AC
DOWNSAMPLING			
143	int downsampledimage[4]; // base address to store the downsampled image	LDI R6	AC ← MBRU ;READ R6 ← AC
144	height_count = image_height/2; // downsampled image_height;	MOV R3,AC	AC←R3
145		RSHIFT1	AC← AC>>1
146		SW X5,AC	AR← MBRU DR ← AC WRITE
147	x = 0;	CLAC	AC←0
148		MOV AC,R4	R4←AC
149	int y = 0;	MOV AC,R5	R5←AC
150	int width_count = image_width/2; // dsimage_width;	MOV R2,AC	AC←R2
151		RSHIFT1	AC← AC>>1
152		SW X12,AC	AR← MBRU DR ← AC WRITE
153	int pixel_value = image[2*y*image_width + 2*x];	MOV R5,AC	AC←R5

154		LSHIFT8	AC <~ AC<<8
155		ADD R4	AC ← AC +R4
156		LSHIFT1	AC ← AC << 1
157		ADD R1	AC ← AC +R1
158		LDAC	AR ← AC; READ IDLE DR ← M(AC) AC ← DR
159		MOV AC,R9	R9←AC
160	downsampledimage[x* (image_width/2) + y] = pixel_value;	MOV R6,AC	AC←R6
161		MOV AC,MAR	MAR←AC
162		INCREMENTAC	AC ← AC+1
163		MOVAC,R6	R6 ← AC
164		MOV R9,AC	AC ← R9
165		MOV AC,MDR	MDR ←AC WRITE
166	y += 1; // moving to next pixel	MOV R5,AC	AC←R5
167		INCREMENT AC	AC←AC+1
168		MOV AC,R5	R5←AC
169	width_count -= 1;	LD X12,R9	MAR ← MBRU;READ IDLE DR ← M(X12) AC ← DR R9 ← AC
170		MOV R9,AC	AC←R9
171		DECREMENT AC	AC←AC-1
172		STAC	DR ← AC WRITE
173	while (width_count > 0): repeat from step 153	JMPNZ 153	AC ← IM(t) PC ← AC PC ←PC+1

174	x += 1; // moving to next row	MOV R4,AC	AC←R4
175		INCREMENT AC	AC←AC+1
176		MOV AC,R4	R4←AC
177	height_count -= 1;	LD X5,R9	MAR ← MBRU; READ IDLE DR ← M(X5) AC ← DR R9 ← AC
178		MOV R9,AC	AC←R9
179		DECREMENT AC	AC←AC-1
180		STAC	DR ← AC WRITE
181	while (width_count > 0): repeat from step 149	JMPNZ 149	AC ← MBRU PC ← AC
182	Complete =1	DONE	DONE

ISA

Instruction	Micro instruction	Break down of microinstruction
FETCH	FETCH1	MBRU ← IRAM[PC]; FETCH
	FETCH2	PC←PC+1
NOOP		IDLE
CLAC		AC←0,Z=1
JUMP	JUMP1	AC← MBRU
	JUMP2	PC←AC
JMPZ	JMPZN1 (Z = 0)	IDLE
	JMPZY1 (Z = 1)	AC ← MBRU
	JMPZY2 (Z = 1)	PC← AC
JMPNZ	JMPNZY1 (Z = 1)	IDLE
	JMPNZN1 (Z = 0)	AC ← MBRU
	JMPNZN2 (Z = 0)	PC← AC
LDAC	LDAC 1	AR ← AC; READ
	LDAC 2	IDLE
	LDAC 3	DR ← M(AC)
	LDAC 4	AC ← DR
LDX1R1	LDX1R1 1	MAR ← MBRU; READ
	LDX1R1 2	IDLE
	LDX1R1 3	DR ← M(X1)
	LDX1R1 4	AC ← DR
	LDX1R1 5	R1 ← AC
LDX2R2	LDX2R2 1	MAR ← MBRU; READ

	LDX2R2 2	IDLE
	LDX2R2 3	DR ← M(X2)
	LDX2R2 4	AC ← DR
	LDX2R2 5	R2 ← AC
LDX3R3	LDX3R3 1	MAR ← MBRU; READ
	LDX3R3 2	IDLE
	LDX3R3 3	DR ← M(X2)
	LDX3R3 4	AC ← DR
	LDX3R3 5	R3 ← AC
LDX12R9	LDX12R9 1	MAR ← MBRU; READ
	LDX12R9 2	IDLE
	LDX12R9 3	DR ← M(X12)
	LDX12R9 4	AC ← DR
	LDX12R9 5	R9 ← AC
LDX5R9	LDX5R9 1	MAR ← MBRU; READ
	LDX5R9 2	IDLE
	LDX5R9 3	DR ← M(X5)
	LDX5R9 4	AC ← DR
	LDX5R9 5	R9 ← AC
LDX4R6	LDX4R6 1	MAR ← MBRU;READ
	LDX4R6 2	IDLE
	LDX4R6 3	DR ← M(X4)
	LDX4R6 4	AC ← DR
	LDX4R6 5	R6 ← AC
SWX5R3	SWX5R3 1	AC←R3
	SWX5R3 2	AR← MBRU
	SWX5R3 3	DR ← AC ;WRITE
SWX12R2	SWX12R2 1	AC←R2

	SWX12R2 2	AR← MBRU
	SWX12R2 3	DR←AC; WRITE
SWX12AC	SWX12AC 1	AR← MBRU
	SWX12AC 2	DR ←AC ;WRITE
SWX5AC	SWX5AC 1	AR← MBRU
	SWX5AC 2	DR ← AC ;WRITE
STAC		DR ← AC ; WRITE
MOVACR1		R1←AC
MOVACR2		R2←AC
MOVACR3		R3←AC
MOVACR4		R4←AC
MOVACR5		R5←AC
MOVACR6		R6←AC
MOVACR7		R7←AC
MOVACR8		R8←AC
MOVACR9		R9←AC
MOVR1AC		AC←R1
MOVR2AC		AC←R2
MOVR3AC		AC←R3
MOVR4AC		AC←R4
MOVR5AC		AC←R5
MOVR6AC		AC←R6
MOVR7AC		AC←R7
MOVR8AC		AC←R8
MOVR9AC		AC←R9
MOVACMAR		MAR←AC
MOVACMDR		MDR ←AC ; WRITE
ADDR1		AC←AC+R1
ADDR2		AC←AC+R2

ADDR4		AC←AC+R4
ADDR5		AC←AC+R5
ADDR6		AC←AC+R6
ADDR7		AC←AC+R7
LSHIFT1		AC ← AC<<1
RSHIFT1		AC ← AC>>1
LSHIFT8		AC ← AC<<8
INCREMENTPC		PC←PC+1
INCREMENTAC		AC←AC+1
DECREMENTAC		AC←AC-1
LDI R2	LDIR2 1	AC← MBRU
	LDIR2 2	R2 ← AC
LDI R3	LDIR3 1	AC ← MBRU
	LDIR3 2	R3 ← AC
LDI R1	LDIR1 1	R1<-- MBRU
LDI R6	LDIR6 1	R6 <-- MBRU
DONE	DONE	

1. INSTRUCTION SET

1.1. PROGRAM CONTROL

~~**START/ INITIALIZE**~~

~~1. PC ß 0~~

~~2. IR ß 0~~

FETCH

1. AR ← PC
2. DR ← M , PC ← PC+1
3. IR ← DR , AR ←PC

NOP

IDLE processor

CLAC

1. AC ← 0 , Z=1

ENDOP

End all operations

JUMP INSTRUCTIONS

JUMP

1. READ
2. AC ← IM(t)
3. PC ← AC

JMPZ

1. READ
2. AC ← IM(t)
3. PC ← AC
4. PC ← PC+1
5. READ

JMPNZ

1. AC ← IM(t)
2. PC ← AC
3. PC ← PC+1

1.2. LOAD AND STORE INSTRUCTIONS

LDIAC

1. MEM READ
2. AC ← IM(t)
3. PC ← PC+1

LDAC

1. AC ← AC
2. READ
3. DR ← M(AC)
4. AC ← DR

LDX1R1

1. MAR ← X1
2. READ
3. DR ← M(X1)
4. AC ← DR
5. R1 ← AC

LDX2R2

1. MAR ← X2
2. READ
3. DR ← M(X2)
4. AC ← DR
5. R2 ← AC

LDX3R3

1. MAR ← X3
2. READ
3. DR ← M(X3)
4. AC ← DR
5. R3 ← AC

LDX12R9

1. MAR ← X12
2. READ
3. DR ← M(X12)
4. AC ← DR
5. R9 ← AC

LDX5R9

1. MAR ← X5
2. READ

3. DR ← M(X12)

4. AC ← DR

5. R9 ← AC

LDX4R6

1. MAR ← X4

2. READ

3. DR ← M(X4)

4. AC ← DR

5. R6 ← AC

SWX5R3

1. AC ← R3

2. AR ← X5

3. DR ← AC

4. WRITE

SWX12R2

1. AC ← R2

2. AR ← X12

3. DR ← AC

4. WRITE

SWX12AC

1. AR ← X12

2. DR ← AC

3. WRITE

SWX5AC

1. AR ← X5

2. DR ← AC

3. WRITE

STAC

1. DR ← AC

2. WRITE

1.3. MOVE INSTRUCTIONS

MOVACR1

1. R1 ← AC

MOVACR2

1. R2 ß AC

MOVACR3

1. R3 ß AC

MOVACR4

1. R4 ß AC

MOVACR5

1. R5 ß AC

MOVACR6

1. R6 ß AC

MOVACR7

1. R7 ß AC

MOVACR8

1. R8 ß AC

MOVACR9

1. R9 ß AC

MOVR1AC

1. AC ß R1

MOVR2AC

1. AC ß R2

MOVR3AC

1. AC ß R3

MOVR4AC

1. AC ß R4

MOVR5AC

1. AC ß R5

MOVR6AC

1. AC ß R6

MOVR7AC

1. AC ß R7

MOVR8AC

1. AC ß R8

MOVR9AC

1. AC ß R9

MOVACMAR

1. MAR ß AC

MOVACMDR

1. MDR ß AC
2. WRITE

MOVAC

1. PC ß AC

2. AR ß PC

1.4. ARITHMETIC AND LOGICAL OPERATIONS

ALU BASED

ADDR1

1. AC ß AC+R1

ADDR2

1. AC ß AC+R2

ADDR4

1. AC ß AC+R4

ADDR5

1. AC ß AC+R5

ADDR6

1. AC ß AC+R6

ADDR7

1. AC ß AC+R7

MULR4

1. AC ß AC*R4

MULR5

1. AC ß AC*R5

LSHIFT

1. AC ß AC<< 1

LSHIFT5

1. AC ß AC<< 5

RSHIFT

1. AC ß AC>> 1

DEDICATED ADDER BASED

INCREMENT PC

1. PC ß PC+1

INCREMENT AC

1. AC ß AC+1

DECREMENT AC

1. AC ß AC-1