// 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 —----------------ADDRESS

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

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  IDLE  DR ← M(X1)  AC ← DR  R1 ← AC |
| 3 | int image\_width = 4; | LD X2,R2 | MAR ← X2 ;READ  IDLE  DR ← M(X2)  AC ← DR  R2 ← AC |
| 4 | int image\_height = 4; | LD X3,R3 | MAR ← X3 ;READ  IDLE  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 R4AC | AC ←R4 |
| 11 | LSHIFT8 | AC ← AC<<8 |
| 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←X12  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<<8 |
| 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<<8 |
| 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 ← X12 ; 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 ← 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 | 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<<8 |
| 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  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 ← 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 R4,AC | AC ←R4 |
| 80 | LSHIFT8 | AC ← AC<<8 |
| 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←X5  DR ← AC  WRITE |
| 89 | int c = image[x\*image\_width + image\_width + y]; | MOV R4,AC | AC ←R4 |
| 90 | LSHIFT8 | AC ← AC<<8 |
| 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 | a = b/2; | MOV R7,AC | AC←R7 |
| 102 | RSHIFT1 | AC ← AC >> 1 |
| 103 | MOV AC,R6 | R6←AC |
| 104 | b = c\*2; | MOV R8,AC | AC←R8 |
| 105 | LSHIFT1 | 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  IDLE  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 | RSHIFT1 | AC ← AC >> 1 |
| 119 | RSHIFT1 | AC ← AC >> 1 |
| 120 | MOV AC,R9 | R9 ←AC |
| 121 | image[x\*image\_width + y] = new\_pixel; | MOV R4,AC | AC ←R4 |
| 122 | LSHIFT8 | AC ← AC<<8 |
| 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  IDLE  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  IDLE  DR ← M(X4)  AC ← DR  R6 ← AC |
| 137 | height\_count = image\_height/2;  // downsampled image\_height; | MOV R3,AC | AC←R3 |
| 138 | RSHIFT1 | 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 | RSHIFT1 | AC← AC>>1 |
| 145 | SW X12,AC | AR←X12  DR ← AC  WRITE |
| 146 | int pixel\_value = image[2\*y\*image\_width + 2\*x]; | MOV R5,AC | AC←R5 |
| 147 | LSHIFT8 | AC <~ AC<<8 |
| 148 | ADD R4 | AC ← AC +R4 |
| 149 | LSHIFT1 | AC ← AC << 1 |
| 150 | ADD R1 | AC ← AC +R1 |
| 151 | LDAC | AR ← AC; READ  IDLE  DR ← M(AC)  AC ← DR |
| 152 | MOV AC,R9 | R9←AC |
| 153 | downsampledimage[x\* (image\_width/2) + y] = pixel\_value; | MOV R4,AC | AC←R4 |
| 154 | RSHIFT1 | AC← AC>>1 |
| 155 | LSHIFT8 | AC ← AC<<8 |
| 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  IDLE  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  IDLE  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 |

| Memory address | Instruction | Micro instruction | Break down of microinstruction |
| --- | --- | --- | --- |
|  | FETCH | FETCH1 | MBRU ← IRAM[PC]; FETCH |
|  | FETCH2 | PC←PC+1 |
|  | FETCH3 | CU←MBRU |
|  | NOOP |  | IDLE |
|  | CLAC |  | AC←0,Z=1 |
|  | JUMP | JUMP1 | READ |
|  | JUMP2 | AC←IM(t) |
|  | JUMP3 | PC←AC |
|  | JMPZ | JMPZ1 |  |
|  | JMPZ2 |  |
|  | JMPZ3 |  |
|  | JMPNZ | JMPNZY1 |  |
|  | JMPNZY1 |  |
|  | JMPNZY1 |  |
|  | LDAC | LDAC 1 | AR ← AC; READ |
|  | LDAC 2 | IDLE |
|  | LDAC 3 | DR ← M(AC) |
|  | LDAC 4 | AC ← DR |
|  | LDX1R1 | LDX1R1 1 | MAR ← X1; READ |
|  | LDX1R1 2 | IDLE |
|  | LDX1R1 3 | DR ← M(X1) |
|  | LDX1R1 4 | AC ← DR |
|  | LDX1R1 5 | R1 ← AC |
|  | LDX2R2 | LDX2R2 1 | MAR ← X2; READ |
|  | LDX2R2 2 | IDLE |
|  | LDX2R2 3 | DR ← M(X2) |
|  | LDX2R2 4 | AC ← DR |
|  | LDX2R2 5 | R2 ← AC |
|  | LDX3R3 | LDX3R3 1 | MAR ← X2; READ |
|  | LDX3R3 2 | IDLE |
|  | LDX3R3 3 | DR ← M(X2) |
|  | LDX3R3 4 | AC ← DR |
|  | LDX3R3 5 | R2 ← AC |
|  | LDX12R9 | LDX12R9 1 | MAR ← X12 ; READ |
|  | LDX12R9 2 | IDLE |
|  | LDX12R9 3 | DR ← M(X12) |
|  | LDX12R9 4 | AC ← DR |
|  | LDX12R9 5 | R9 ← AC |
|  | LDX5R9 | LDX5R9 1 | MAR ← X5; READ |
|  | LDX5R9 2 | IDLE |
|  | LDX5R9 3 | DR ← M(X5) |
|  | LDX5R9 4 | AC ← DR |
|  | LDX5R9 5 | R9 ← AC |
|  | LDX4R6 | LDX4R6 1 | MAR ← X4 ;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←X5 |
|  | SWX5R3 3 | DR ← AC ;WRITE |
|  | SWX12R2 | SWX12R2 1 | AC←R2 |
|  | SWX12R2 2 | AR←X12 |
|  | SWX12R2 3 | DR←AC; WRITE |
|  | SWX12AC | SWX12AC 1 | AR←X12 |
|  | SWX12AC 2 | DR ←AC ;WRITE |
|  | SWX5AC | SWX5AC 1 | AR←X5 |
|  | SWX5AC 2 | DR ← AC ;WRITE |
|  | STAC |  | DR ← AC ; WRITE |
|  | MOVACR1 |  | R1←AC |
|  | MOVACR2 |  | R2←AC |
|  | MOVACR3 |  | R3←AC |
|  | MOVACR4 |  | R4←AC |
|  | MOVACR5 |  | R6←AC |
|  | MOVACR6 |  | R7←AC |
|  | MOVACR7 |  | R8←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 |

# **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. R6ßAC

#### **SWX5R3**

1. ACßR3

2. AR ß X5

3. DR ß AC

4. WRITE

#### **SWX12R2**

1. ACßR2

2. AR ß X12

3. DR ß AC

4. WRITE

#### **SWX12AC**

1. AR ß X12

2. DR ß AC

3. WRITE

#### **SWX5AC**

1. AR ß X5

2. DR ß AC

3. WRITE

#### **STAC**

1. DR ß AC

2. WRITE

## **1.3.** **MOVE INSTRUCTIONS**

#### **MOVACR1**

1. R1 ß 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