Lab 4: Image Blurring

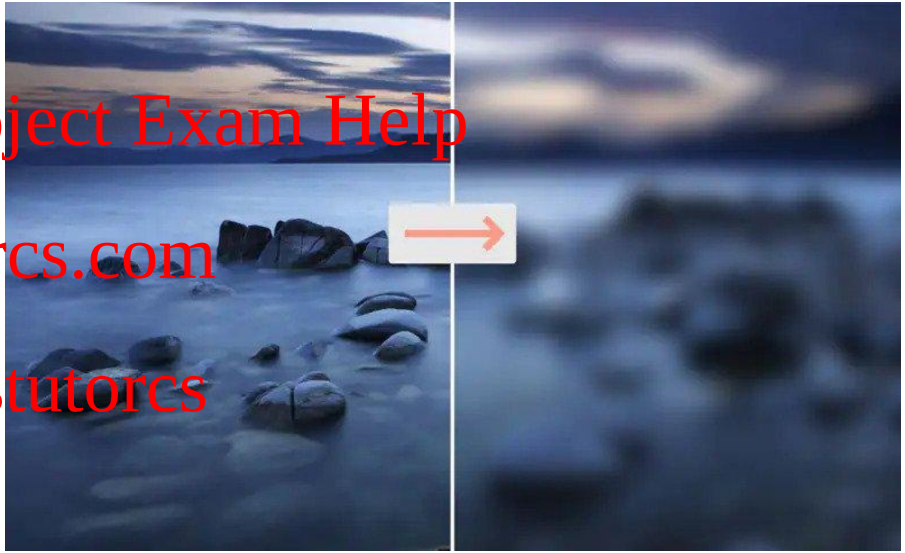
CMPUT 229
University of Alberta

Lab Requirements

- Function calls and register conventions
- Loading and storing from memory
- Loops and control flow
- Bit manipulation

Background

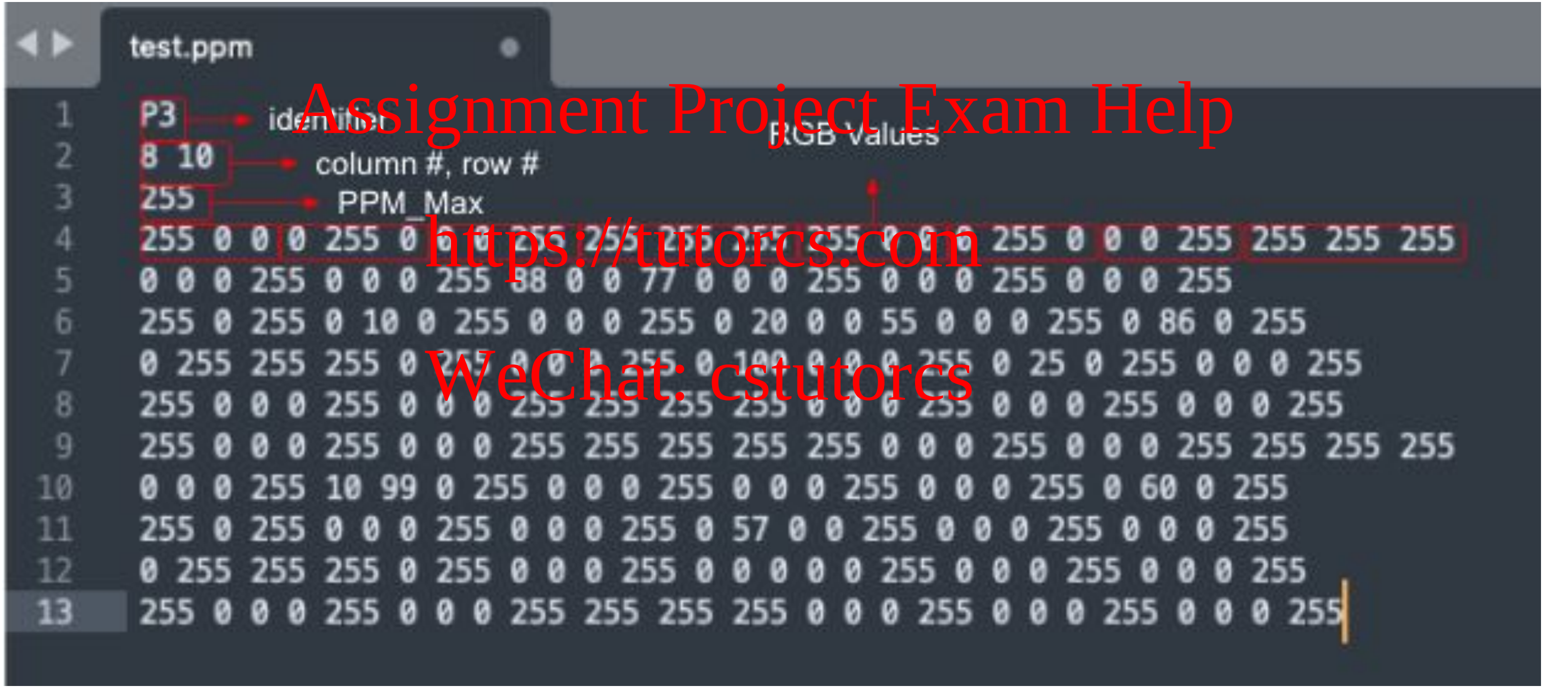
- Common technique to hide key information
- Distort the detail of an image to make it less clear



Box Blur

$$\frac{1}{49}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

PPM format:


test.ppm
identifier
column #, row #
PPM_Max
RGB Values
P3
8 10
255
255 0 0 0 255 0 0 0 255 255 255 255 255 0 0 0 255 0 0 0 255 255 255 255
0 0 0 255 0 0 0 255 88 0 0 77 0 0 0 255 0 0 0 255 0 0 0 255
255 0 255 0 10 0 255 0 0 0 255 0 20 0 0 55 0 0 0 255 0 86 0 255
0 255 255 255 0 255 0 0 255 0 100 0 0 0 255 0 25 0 255 0 0 0 255
255 0 0 0 255 0 0 0 255 255 255 255 0 0 0 255 0 0 0 255 0 0 0 255
255 0 0 0 255 0 0 0 255 255 255 255 255 0 0 0 255 0 0 0 255 255 255 255
0 0 0 255 10 99 0 255 0 0 0 255 0 0 0 255 0 0 0 255 0 60 0 255
255 0 255 0 0 0 255 0 0 0 255 0 57 0 0 255 0 0 0 255 0 0 0 255
0 255 255 255 0 255 0 0 0 255 0 0 0 0 0 255 0 0 0 255 0 0 0 255
255 0 0 0 255 0 0 0 255 255 255 255 0 0 0 255 0 0 0 255 0 0 0 255

The Assignment

Part 1:

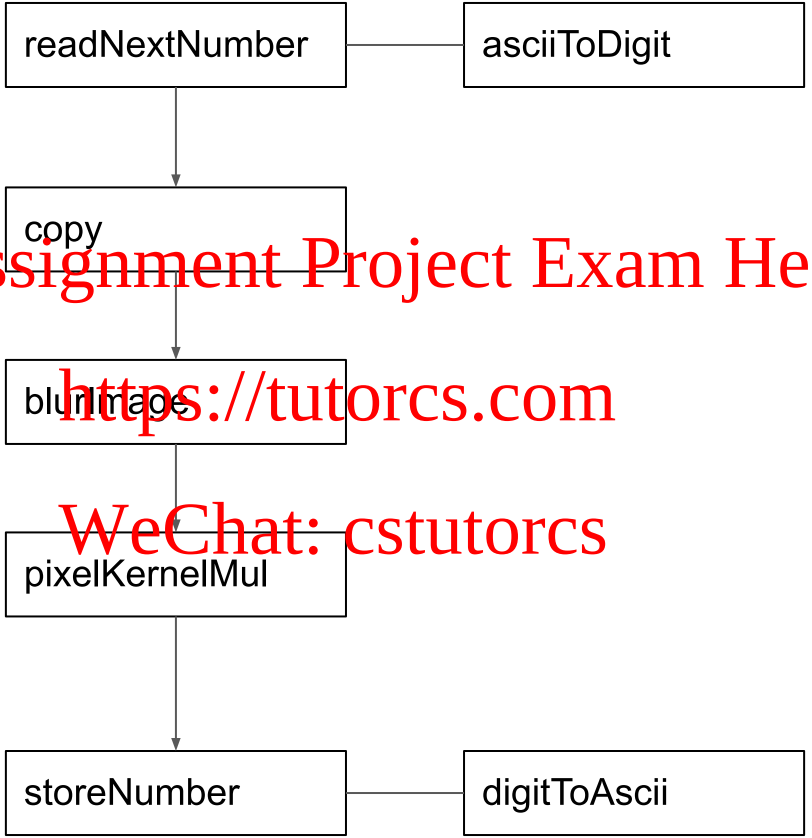
- digitToAscii
- copyImage
- asciiToDigit

Part 2:

- readNextNumber
- storeNumber

Part 3:

- pixelKernelMul
- blurImage


readNextNumber
asciiToDigit
copy
blurImage
pixelKernelMul
storeNumber
digitToAscii

Subroutines

digitToAscii:

This function returns the ASCII value of the digit.

Arguments:

a0: a single digit represented as an integer, between 0 and 9

Return:

a0: the ASCII value of the digit, between 48 (0x30) and 57 (0x39)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Subroutines

copy:

This function copies all the RGB values to another address.

Arguments:

a0: address of the start of RGB values (where to copy from)

a1: address of the start of where to copy to

a2: length in words

Subroutines

asciiToDigit:

This function returns the digit for the given ASCII value

Arguments:

a0: the ASCII value of the digit, between 48 (0x30) and 57 (0x39)

Return:

a0: a single digit represented as an integer, between 0 and 9

Subroutines

readNextNumber:

It reads a string of ASCII characters and converts the first number it finds into an integer. It is guaranteed to only have ASCII numbers between 0 and 255. The function skips any whitespace before the number, then reads until it encounters a whitespace character including:"space" or "\n" or "\r" or "\t"

Arguments:

a0: the current address to start reading

Return:

a0: the address to start reading the next number

a1: the number represented as an integer

Example:

Arguments:
a0: 0x10010000

Returns:	
a0	0x10010003
a1	0x00000020 (32 in decimal)

3 in ASCII

2 in ASCII

space in ASCII
(stop reading)

next byte to read

Address	Value (+0)	Value (+4)
0x10010000	0x20203233	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000

Example:

Arguments:
a0: 0x10010000

Returns:	
a0	0x10010005
a1	0x00000020 (32 in decimal)

space in ASCII

space in ASCII

3 in ASCII

2 in ASCII

next byte to read

Address	Value (+0)	Value (+4)
0x10010000	0x32332020	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000

NULL in ASCII
(stop reading)

Example:

Arguments:
a0: 0x10010000

Returns:	
a0	0x10010005
a1	0x00000001 (1 in decimal)

0 in ASCII

0 in ASCII

0 in ASCII

1 in ASCII

next byte to read

Address	Value (+0)	Value (+4)
0x10010000	0x31303030	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000

NULL in ASCII
(stop reading)

Example:

Arguments:
a0: 0x10010000

Returns:	
a0	0x10010005
a1	0x00000000 (0 in decimal)

0 in ASCII

0 in ASCII

0 in ASCII

0 in ASCII

next byte to read

Address	Value (+0)	Value (+4)
0x10010000	0x30303030	0x0000000
0x10010020	0x00000000	0x0000000
0x10010040	0x00000000	0x0000000
0x10010060	0x00000000	0x0000000
0x10010080	0x00000000	0x0000000
0x100100a0	0x00000000	0x0000000

NULL in ASCII
(stop reading)

Subroutines

storeNumber:

This function converts the integer(at most 3 digits) to ASCII and then store their ASCII to the address. Store the leftmost digit first, consider using the function digitToAscii.

Arguments:

a0: the number represented as an integer

a1: address for the integer to be stored

Return:

a0: the next address that's available to be stored

Example:

Arguments:
a0: 0x000000ff (255 in decimal)
a1: 0x10010000

Return:
a0: 0x10010003

Address	Value (+0)	Value (+4)
0x10010000	0x00353532	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000

Subroutines

pixelKernelMul:

This function calculate the average for each of the R, G, B values and store the value.

Arguments:

a0: address of the start of RGB values, store the calculated RGB values in this region (a0 is the base address)

a1: address of the start of the copy of RGB values

a2: row # of the current pixel to blur

a3: col # of the current pixel to blur

a4: total row (may not be used)

a5: total col

The pixels in memory:

R value (18 in dec)

G value (0)

B value (255)

Pixel 1:

Pixel 2:

Pixel 3:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00ff0012	0x001100ff	0x00ff00ff	0x00ff0012
0x10010020	0x00ff00ff	0x00ff0012	0x001100ff	0x00ff00ff
0x10010040	0x001100ff	0x00ff00ff	0x00ff0012	0x001100ff
0x10010060	0x00ff0012	0x001100ff	0x00ff00ff	0x00ff0012
0x10010080	0x00ff00ff	0x00ff0012	0x001100ff	0x00ff00ff
0x100100a0	0x001100ff	0x00ff00ff	0x00ff0012	0x001100ff

Null (placeholder)

Big-endian:

```
+---+---+---+---+---+---+-----------+
| 0 | 1 | 2 | 3 | 4 | 5 |    ...    | 255 |
+---+---+---+---+---+---+-----------+
| R | G | B |Nul| R | G |    ...    | Nul |
+---+---+---+---+---+---+-----------+
<--------------->
        |
     1 pixel
```

Original:

3	5	7	8	6	7	1	5	2	2
7	5	3	2	1	5	9	8	7	6
6	7	2	5	9	10	1	5	1	1
6	1	4	6	6	1	6	8	9	1
5	1	3	8	0	1	0	1	7	6
1	9	7	5	5	6	1	9	8	8
6	8	5	4	3	1	5	7	8	0
1	0	10	9	8	9	7	5	0	1
7	6	9	8	6	6	6	1	0	1
0	1	8	7	4	3	9	0	1	0

new[x][y] = average of all of the surrounding 49 elements

$$\begin{aligned} new[x][y] = (&old[x-3][y-3] + old[x-2][y-3] + old[x-1][y-3] + old[x][y-3] \\ &+old[x+1][y-3] + old[x+2][y-3] + old[x+3][y-3] \\ &+old[x-3][y-2] + old[x-2][y-2] + old[x-1][y-2] + old[x][y-2] \\ &+old[x+1][y-2] + old[x+2][y-2] + old[x+3][y-2] \\ &+old[x-3][y-1] + old[x-2][y-1] + old[x-1][y-1] + old[x][y-1] \\ &+old[x+1][y-1] + old[x+2][y-1] + old[x+3][y-1] \\ &+old[x-3][y] + old[x-2][y] + old[x-1][y] + old[x][y] \\ &+old[x+1][y] + old[x+2][y] + old[x+3][y] \\ &+old[x-3][y+1] + old[x-2][y+1] + old[x-1][y+1] + old[x][y+1] \\ &+old[x+1][y+1] + old[x+2][y+1] + old[x+3][y+1] \\ &+old[x-3][y+2] + old[x-2][y+2] + old[x-1][y+2] + old[x][y+2] \\ &+old[x+1][y+2] + old[x+2][y+2] + old[x+3][y+2] \\ &+old[x-3][y+3] + old[x-2][y+3] + old[x-1][y+3] + old[x][y+3] \\ &+old[x+1][y+3] + old[x+2][y+3] + old[x+3][y+3])/49 \end{aligned}$$

$$\frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} new[5][5] = (&2+5+9+10+1+5+1+4+6+6+1+6+8+9+3+8 \\ &+0+1+0+1+7+7+5+5+0+1+9+8+5+4 \\ &+3+1+5+7+8+10+9+8+9+7+5+0+9 \\ &+8+6+6+6+1+0)/49 \\ = 5& \end{aligned}$$

Subroutines

blurImage:

This function blurs the image using pixelKernelMul on all possible pixels. It will run PixelKernelMul on all pixels except for the ones on the edges and corners.

Arguments:

a0: kernel size

a1: total rows

a2: total columns

a3: address of the start of RGB values, store the calculated RGB values in this region (a0 is the base address)

a4: address of the start of the copy of RGB values

Pseudo Code


```
const int kernel_radius = 3;
for (int i = kernel_radius; i < row - kernel_radius; i++) {
  for (int j = kernel_radius; j < col - kernel_radius; j++) {
    PixelKernelMul(...);
  }
}
```

Register Conventions in RISC-V

Register conventions are required for this lab. Review your notes on using the stack pointer and register conventions.

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Tips and Notes

- Loops are a very important part of this lab. Review your notes.
- For the simplicity of this lab, we are only blurring the center part of the lab and ignoring the edges and the corners.
- You can make any other helper functions to break down the problem.
- Read carefully about the instructions and examples provided.
- Do not edit any part of common.s
- Do not use any labels used in common.s