

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



# ASSIGNMENT REPORT

## Computer Architecture

**TOPIC: CONVOLUTION OPERATION**

Class : CC08 - HK241

INSTRUCTOR : NGUYEN THIEN AN

Student Code	Name
2352911	Tan Khanh Phong

Ho Chi Minh City, 2024

# Contents

<b>1 Outcomes</b>	<b>2</b>
<b>2 Introduction</b>	<b>2</b>
<b>3 Requirements</b>	<b>4</b>
3.1 Input . . . . .	4
3.2 Output . . . . .	4
3.3 Algorithm . . . . .	4
3.4 Detailed explanation some important part of the code . . . . .	7
3.5 Testcases . . . . .	10
3.6 Results of some testcases . . . . .	10
<b>4 Reference books :</b>	<b>12</b>

## 1 Outcomes

After finishing this assignment, students can proficiently use:

- MARS MIPS simulator.
- Arithmetic & data transfer instructions.
- Conditional branch and unconditional jump instructions.
- Procedures.

## 2 Introduction

In this modern day and age, artificial intelligence has been developing exponentially fast. Rapid improvements in technique and data lead to a remarkable increase in calculations. Its sub-categories, machine learning and deep learning, have reached a point where billions of calculations must be done to compute the millions of parameters in a model.

Convolutional neural network (CNN) is one of the most widely used type of network used in deep learning. Its usage can be seen in a wide range of applications, particularly in fields that involve image and video analysis. However advanced a CNN model maybe, it always stems from matrix computations and most notably, convolution operations.

The convolution operation involves sliding a filter, also known as a kernel, across the input image. This filter is a small matrix of weights that is applied to a local region of the input matrix, producing a single value in the output feature map. The process is repeated across the entire matrix, allowing the network to learn various features of the matrix. In the case of image analysis, these features can include edges, textures, and patterns. The mathematical operation performed is essentially a dot product between the filter and the receptive field of the input image.

The process involves taking a small matrix, called a kernel or filter, and sliding it over an input matrix, such as an image. At each position, the dot product of the kernel and the overlapping region of the input matrix is computed, and the result is stored in the output matrix. This process is repeated for all positions of the kernel over the input matrix, effectively blending the kernel with the input to highlight specific features like edges or textures. In advance,

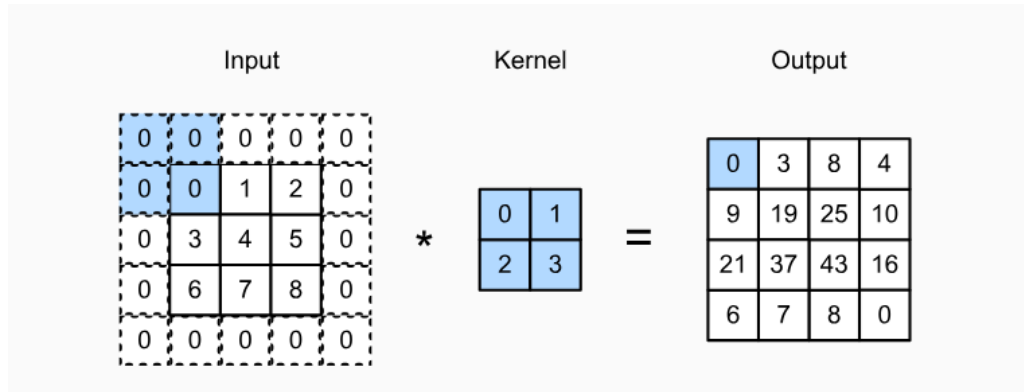


Figure 1: An example of convolution operation

In Figure 1, a 5x5 input matrix and a 2x2 kernel filter have been provided. We first match the kernel onto a corner of the input, in this case, the small 4x4 matrix covered in a large red square. We then calculate the sum of element-wise products of both matrices to get a single value. This value will be the first value of the output matrix. Proceed to slide the kernel from left to right, then up to down to cover every square of the input to get the final result. Notice that the stride here is 1 as the kernel moves 1 column and 1 row at a time.

One tricky issue when applying convolutional layers is that we tend to lose pixels on the perimeter of our image. One straightforward solution to this problem is to add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image. Typically, we set the values of the extra pixels to zero.

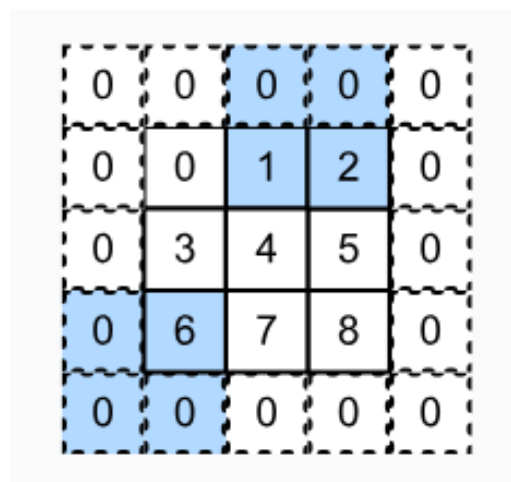


Figure 2: An example of matrix with zero-padding

In Figure 2, applying the padding with value equal to 1 into the 3x3 input matrix will increase its size to 5x5.

## 3 Requirements

In this assignment, your task is to perform convolution operations using MIPS assembly. The requirements are listed in the following sections

### 3.1 Input

The program should be able to receive input from an external input file. It should be named **input\_matrix.txt**. The content is numbers separated by spaces. Both image matrix and kernel matrix are square matrix **m x m** and **n x n**, respectively. Additionally, **input\_matrix.txt** contain 3 rows, the first row will include 4 values which are:

- **N**: The size of the image matrix, with the constraint:

$$3 \leq N \leq 7$$

- **M**: The size of the kernel matrix, with the constraint:

$$2 \leq M \leq 4$$

- **p**: The value of padding, with the constraint:

$$0 \leq p \leq 4$$

- **s**: The value of stride, with the constraint:

$$1 \leq s \leq 3$$

All of the matrices content are **floating-point numbers** rounded to 1 decimal point while the stride and padding are **integer numbers**. However, all of the numbers are presented as floats with 1 digit after the decimal point.

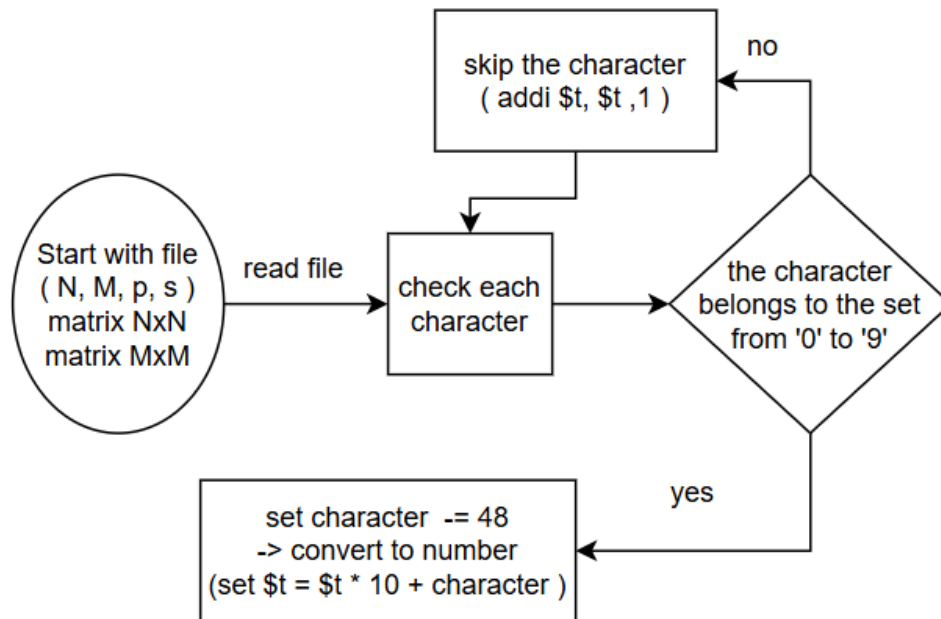
### 3.2 Output

A text file named **output\_matrix.txt** contains the convolution matrix result, also numbers separated by spaces

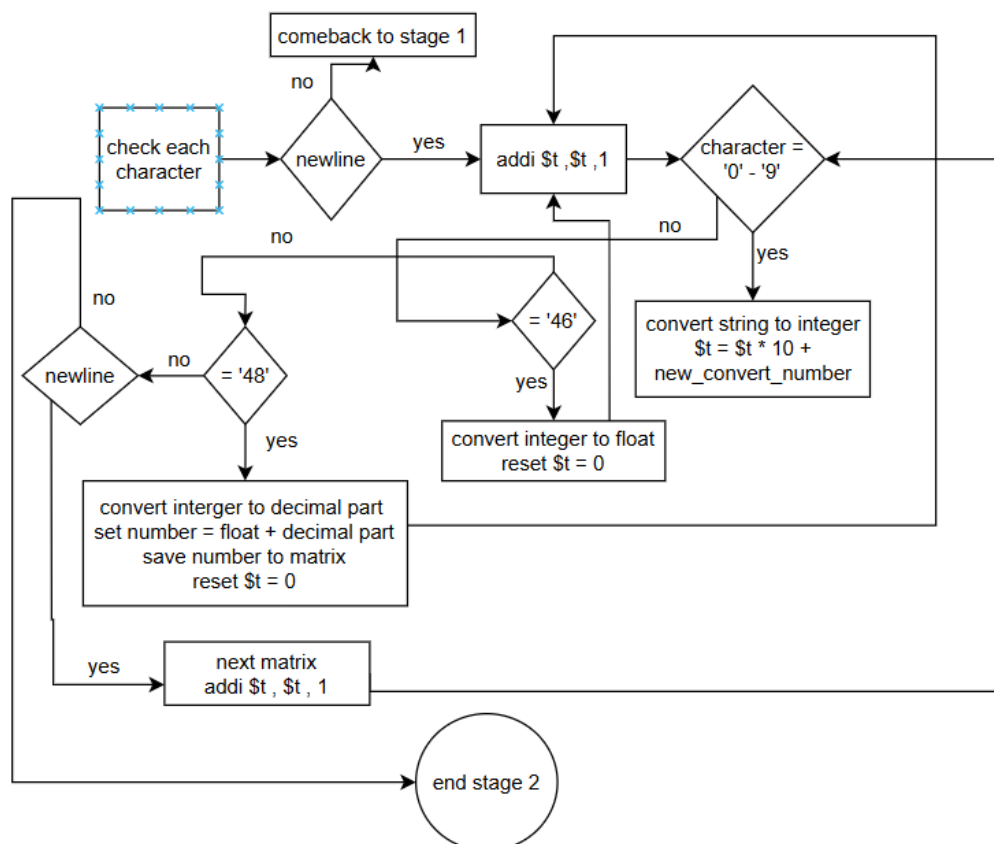
### 3.3 Algorithm

The algorithm is divided into 4 stages :

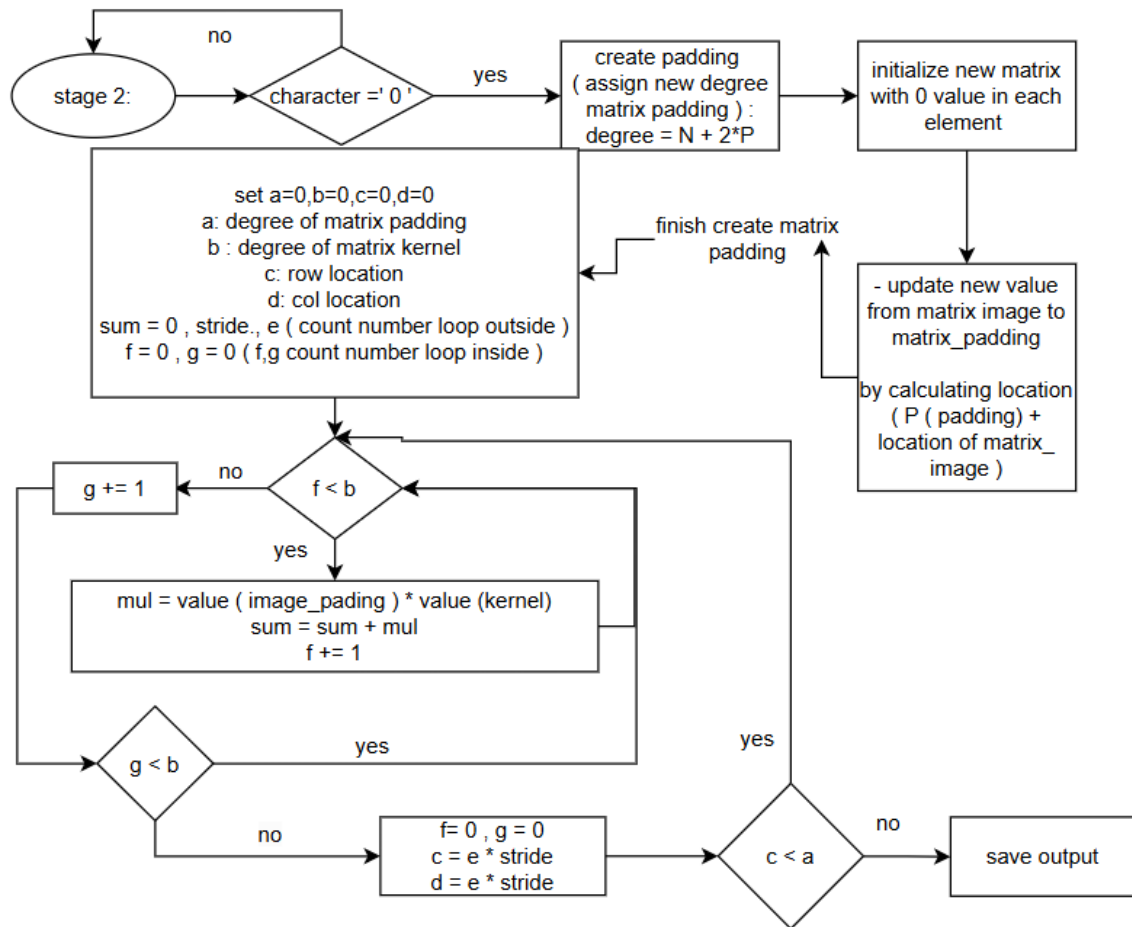
- Stage 1 : read file and get N,M,P,s



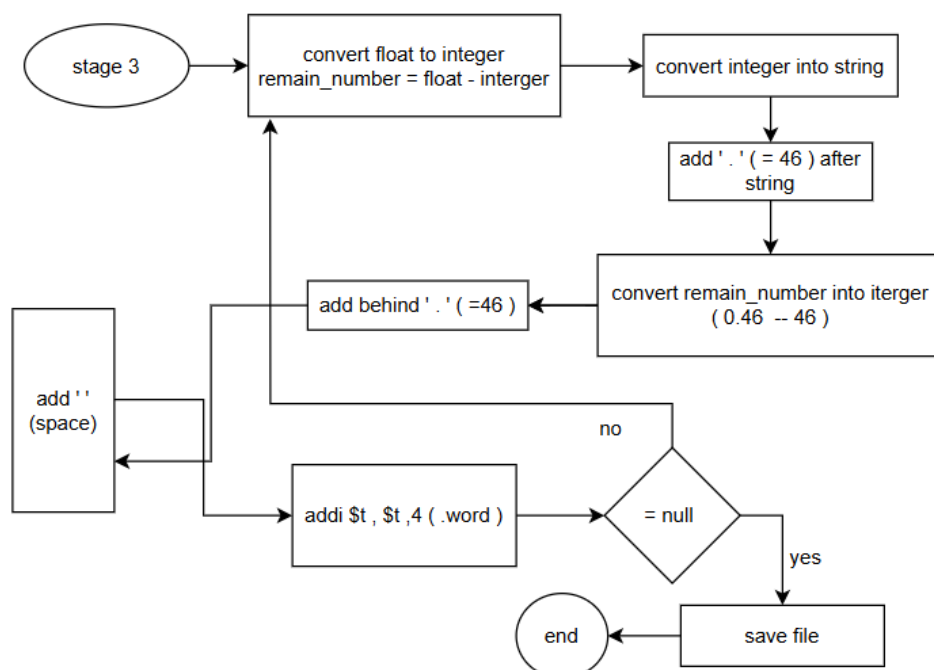
- Stage 2 : save matrix N M



- Stage 3 : create matrix padding and calculate output



- Stage 4 : convert into string and save file



## 3.4 Detailed explanation some important part of the code

### MIPS Reference Sheet

#### Branch Instructions

Instruction	Operation
beq \$s, \$t, label	if ( $\$s == \$t$ ) $pc += i \ll 2$
bgtz \$s, label	if ( $\$s > 0$ ) $pc += i \ll 2$
blez \$s, label	if ( $\$s \leq 0$ ) $pc += i \ll 2$
bne \$s, \$t, label	if ( $\$s \neq \$t$ ) $pc += i \ll 2$

#### Arithmetic and Logical Instructions

Instruction	Operation
add \$d, \$s, \$t	$\$d = \$s + \$t$
addu \$d, \$s, \$t	$\$d = \$s + \$t$
addi \$t, \$s, i	$\$t = \$s + SE(i)$
addiu \$t, \$s, i	$\$t = \$s + SE(i)$
and \$d, \$s, \$t	$\$d = \$s \& \$t$
andi \$t, \$s, i	$\$t = \$s \& ZE(i)$
div \$s, \$t	$lo = \$s / \$t; hi = \$s \% \$t$
divu \$s, \$t	$lo = \$s / \$t; hi = \$s \% \$t$
mult \$s, \$t	$hi:lo = \$s * \$t$
multu \$s, \$t	$hi:lo = \$s * \$t$
add.s	$\$f0 = \$f1 + \$f2$
sub.s	$\$f0 = \$f1 - \$f2$
mul.s	$\$f0 = \$f1 * \$f2$
div.s	$\$f0 = \$f1 / \$f2$

#### Jump Instructions

Instruction	Operation
j label	$pc += i \ll 2$
jal label	$\$31 = pc; pc += i \ll 2$
jalr \$s	$\$31 = pc; pc = \$s$
jr \$s	$pc = \$s$



## Load Instructions

Instruction	Operation
lb \$t, i(\$s)	\$t = SE (MEM [\$s + i]:1)
lbu \$t, i(\$s)	\$t = ZE (MEM [\$s + i]:1)
lh \$t, i(\$s)	\$t = SE (MEM [\$s + i]:2)
lhu \$t, i(\$s)	\$t = ZE (MEM [\$s + i]:2)
lw \$t, i(\$s)	\$t = MEM [\$s + i]:4

## Store Instructions

Instruction	Operation
sb \$t, i(\$s)	MEM [\$s + i]:1 = LB (\$t)
sh \$t, i(\$s)	MEM [\$s + i]:2 = LH (\$t)
sw \$t, i(\$s)	MEM [\$s + i]:4 = \$t

## Data Movement Instructions

Instruction	Operation
mfhi \$d	\$d = hi
mflo \$d	\$d = lo
mthi \$s	hi = \$s
mtlo \$s	lo = \$s

## Comparison Instructions

Instruction	Operation
slt \$d, \$s, \$t	\$d = (\$s < \$t)
sltu \$d, \$s, \$t	\$d = (\$s < \$t)
slti \$t, \$s, i	\$t = (\$s < SE(i))
sltiu \$t, \$s, i	\$t = (\$s < SE(i))
c.eq.s	Compare floating-point values for equality: Set flag = 1 if \$f0 == \$f1
c.lt.s	Compare floating-point values for less than: Set flag = 1 if \$f0 < \$f1
c.le.s	Compare floating-point values for less than or equal: Set flag = 1 if \$f0 <= \$f1

## Directives

Directive	Descriptions
<code>.word</code>	32-bit integer
<code>.float</code>	Floating point IEEE 754 single precision
<code>.space</code>	Uninitialized memory block (byte)
<code>.ascii</code>	Store the string in memory, but do not null-terminate it.
<code>.asciiz</code>	Store the string in memory and null-terminate it.
<code>.align</code>	Align the next datum on a 2n byte boundary.

## Conversion Instructions

Instruction	Operation
<code>cvt.s.w</code>	Convert integer to single-precision floating-point: $\$f0 = \text{float}(\$f1)$
<code>cvt.w.s</code>	Convert single-precision floating-point to integer: $\$f0 = \text{int}(\$f1)$
<code>cvt.d.s</code>	Convert single-precision floating-point to double-precision: $\$f0 = \text{double}(\$f1)$

## Rounding Instructions

Instruction	Operation
<code>round.w.s</code>	Round single-precision floating-point to the nearest integer: $\$f0 = \text{round}(\$f1)$
<code>trunc.w.s</code>	Truncate single-precision floating-point to integer: $\$f0 = \text{trunc}(\$f1)$
<code>floor.w.s</code>	Round single-precision floating-point down to the nearest integer: $\$f0 = \text{floor}(\$f1)$
<code>ceil.w.s</code>	Round single-precision floating-point up to the nearest integer: $\$f0 = \text{ceil}(\$f1)$

## 3.5 Testcases

Testcase 1 :

```
5.0 3.0 0.0 1.0  
1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0  
1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0
```

Testcase 2 :

```
5 3 0 1  
1.2 1.5 2.1 0 0 0 1.0 1.0 1.0 0 0 0 1.0 1.0 1.0 0 0 1.0 1.0 0 0 1.0 1.0 0 0  
1.0 0 1.0 0 1.0 0 1.0 0 1.0
```

Testcase 3 :

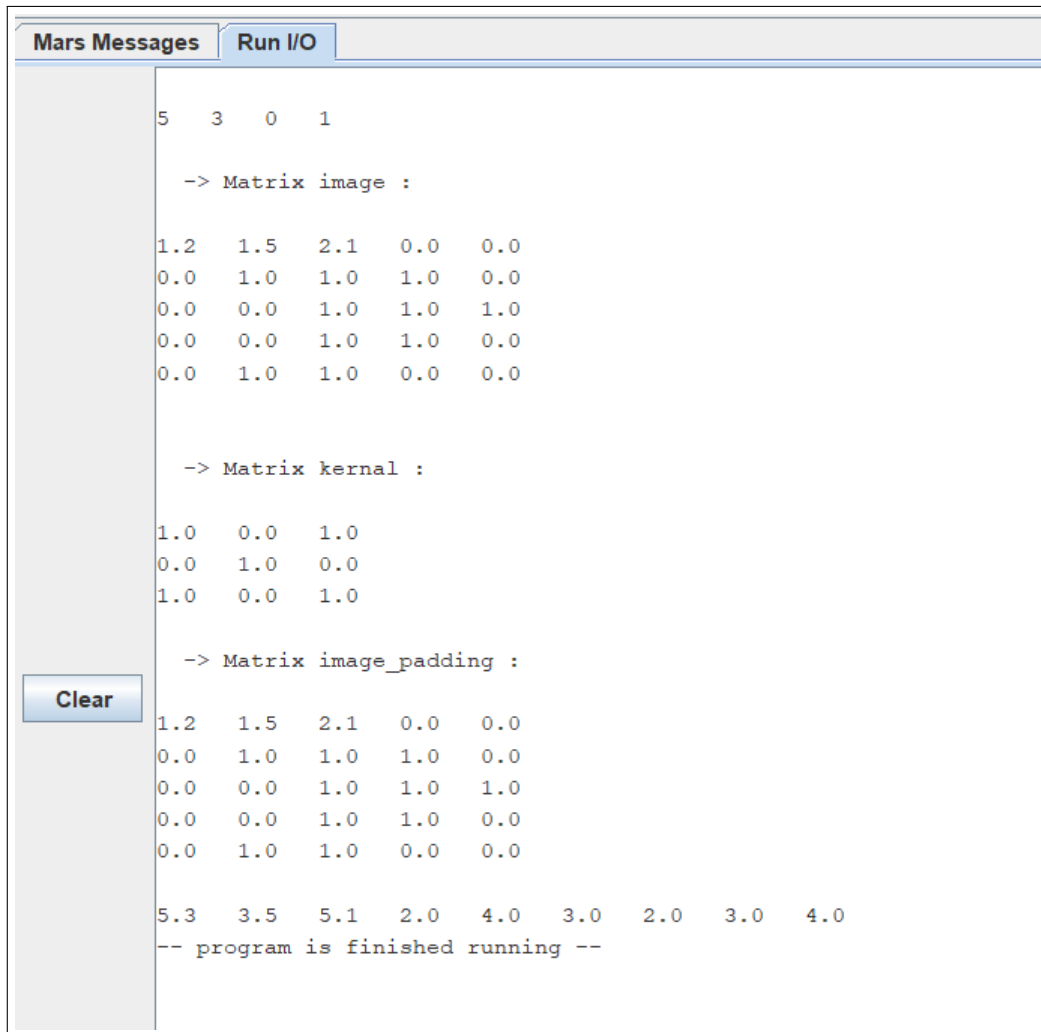
```
2 4 1 2  
-3.0 -4 4.5 6  
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
```

Testcase 4 :

```
8.0 4.0 0 1  
1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0  
0.0 0.0 1.0 1.0 0.0 0.0  
1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0
```

## 3.6 Results of some testcases

## Testcase 1 and testcase 2 :

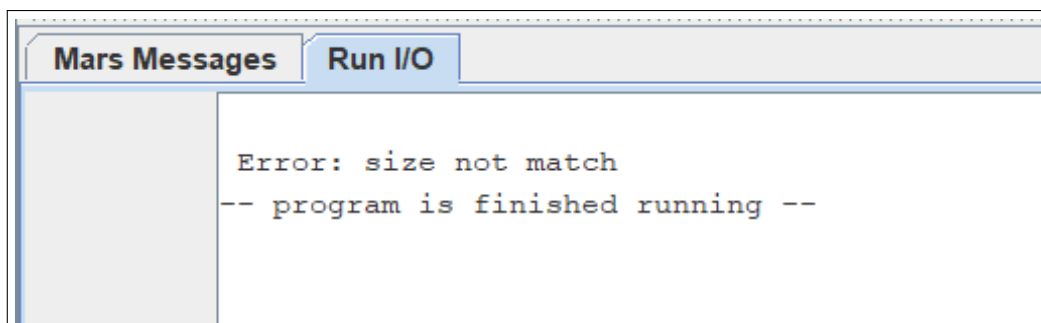


```
Mars Messages Run I/O
5 3 0 1
-> Matrix image :
1.2 1.5 2.1 0.0 0.0
0.0 1.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0 1.0
0.0 0.0 1.0 1.0 0.0
0.0 1.0 1.0 0.0 0.0
-> Matrix kernal :
1.0 0.0 1.0
0.0 1.0 0.0
1.0 0.0 1.0
-> Matrix image_padding :
1.2 1.5 2.1 0.0 0.0
0.0 1.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0 1.0
0.0 0.0 1.0 1.0 0.0
0.0 1.0 1.0 0.0 0.0
5.3 3.5 5.1 2.0 4.0 3.0 2.0 3.0 4.0
-- program is finished running --
```

## Output file :

5.3 3.5 5.1 2.0 4.0 3.0 2.0 3.0 4.0

## Testcase 3 and testcase 4 :



```
Mars Messages Run I/O
Error: size not match
-- program is finished running --
```

Output file :



Error: size not match

#### 4 Reference books :

- **MIPS32 Architecture for Programmers Volume II: The MIPS32 Instruction Set:** Comprehensive documentation on MIPS32 instructions and usage.
- **Computer Organization and Design: The Hardware/Software Interface** by David A. Patterson and John L. Hennessy. A fundamental textbook covering MIPS architecture.
- **MIPS Assembly Language Programming** by Robert Britton. Practical guide for learning and writing MIPS assembly programs.
- **The Essentials of Computer Organization and Architecture** by Linda Null and Julia Lobur. Explains foundational computer architecture concepts, including MIPS.