The Operation & Instruction of MIPS Simulator

Course	Computer Organization and Design
Academic Advisor	Professor Xueqing Lou
Teaching Assistants	
Name	Leming Shen
ID Number	3180103654
College	College of Computer Science and Technology
Major	Software Engineering

April 14, 2020

Zhejiang University Software Report

Course: Computer Organization and Design Type of Experiment: Comprehensive

Software Item: MIPS Simulator

Academic Advisor: Professor Xueqing Lou Teaching Assistants:

Name: Leming Shen Major: Software Engineering ID Number: 3180103654

Experiment Location: Home, Hangzhou Experiment Time: April 14, 2020

CONTENTS

1. Introduction

1.1. Target

1.1.1. MIPS assembly language's assembling procedure implementation.

1.1.2. Machine Codes' disassembling procedure implementation.

1.1.3. MIPS execution monitor.

1.2. Background

1.2.1. Getting know deeper about MIPS assembly language and its execution procedure.

1.2.2. Getting better understand the CPU's working principles.

1.2.3. Implementing instructions and machine codes' mutual transformation.

1.2.4. Simulate to monitor MIPS assembly code execution.

1.3. Significance

1.3.1. A "soft" compiler.

1.3.2. Help us better understand how CPU works and data transformation among memory and registers.

1.4. References

1.4.1. *Computer Organization and Design*, The Hardware/Software Interface (Fifth Edition), *David A. Patterson*, *John L. Hennessy*.

1.4.2. *Logic and Computer Design Fundamentals*, Fifth Edition, *M. Morris Mano*, California State University, Los Angeles, *Charles R. Kime*, University of Wisconsin, Madison, *Tom Martin*, Virginia Tech.

1.4.3. *Computer Systems, A Programmer's Perspective*, Second Edition, Randal E. Bryant, Carnegie Mellon University, David R. O'Hallaron, Carnegie Mellon University and Intel Labs.

1.4.4. *Microsoft Visual C# Step by Step*, Ninth Edition, *John Sharp.*

1.4.5. *Starting Out with C++ Early Objects*, Ninth Edition, Tony Gaddis, Judy Walters,Godfrey Muganda.

1.4.6. *Principles of Data Structures using C and C++*, Vinu V Das, M. E. S. College of Engineering Kuttippuram, Kerala, India.

1.4.7. *The C# Programming Language*, Fourth Edition, Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, Peter Golde.

1.4.8. *Essential Algorithms, A Practical Approach to Computer Algorithms*, Rod Stephens

1.4.9. *Algorithmic Problem Solving*, Roland Backhouse, University of Nottingham

1.4.10. *Introduction to Algorithms*, Third Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

1.4.11. *Algorithm Design*, Jon Kleinberg, Cornell University.

1.4.12. *Introduction to Programming with C++*, Third Edition, Y. Daniel Liang.

1.4.13. *Software Engineering, A Practitioner's Approach*, Eighth Edition, Roger S. Pressman, Bruce R. Maxim.

1.4.14. *Software Engineering, Principles and Practice*, Third Edition, Hans van Vliet.

1.5. Function

1.5.1. Convert MIPS assembly language to machine code.

1.5.2. Convert machine code to MIPS assembly language.

1.5.3. Execute MIPS assembly code. (Comment not supported)

2. Environment & Computer Requirement

2.1. Hardware Support

- A computer that has a 1.8 GHz or faster processor (dual-core or better recommended).
- 2 GB RAM (4 GB RAM recommended, add 512 MB if running in a virtual machine).
- 10 GB of available hard disk space after installing Visual Studio 5400 RPM hard-disk drive (SSD recommended).
- A video card that supports a 1024 × 768 or higher resolution display.
- Internet connection to download software or chapter examples.

2.2. Operating System

- Windows 10 (Home, Professional, Education, or Enterprise) version 1803 or higher.
- Windows 10 download: https://www.microsoft.com/zh-cn/windows/
- With Developers Mode.

2.3. Software Support

- The most recent build of Visual Studio Community 2019, Visual Studio Professional 2019, or Visual Studio Enterprise 2017 (make sure that you have installed any updates). As a minimum, you should select the following workloads when installing Visual Studio 2019:

- ✓ Universal Windows Platform development
- ✓ .NET desktop development
- ✓ ASP.NET and web development
- ✓ Azure development
- ✓ Data storage and processing
- ✓ .NET Core cross-platform development

➢ Microsoft Visual Studio 2019 download: https://visualstudio.microsoft.com/

2.4. Alternative Option

If your computer is not Windows Operating System or your computer is not equipped with Microsoft Visual Studio 2019, you can directly watch the video named "Instruction Video.mp4" in the project folder.

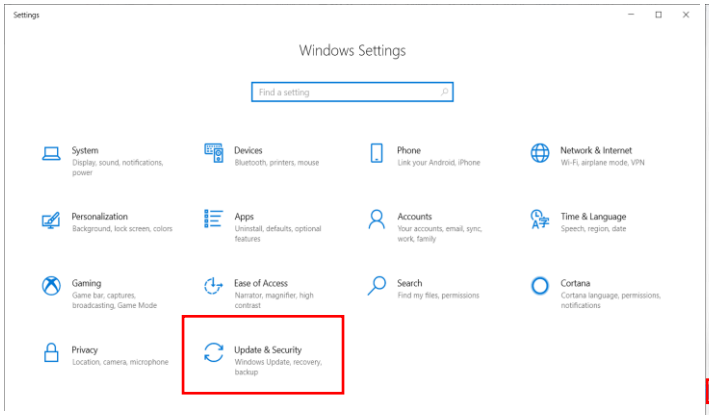
3. Software Use Procedure

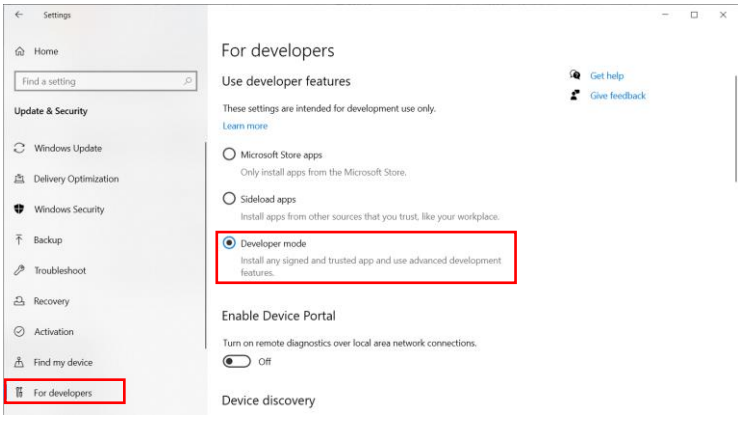
3.1. Start Up

3.1.1. The MIPS Simulator is an application what we called UWP (Uniform Windows Platform), its style is brand new similar to Windows10 Operating System, which enjoys a high level of clean and beauty.

3.1.2. To get started with Visual Studio 2019 and UWP, you need to enable Developer Mode for Windows 10. Steps are as follows:

- ➢ Press "Windows" key or left click on the left bottom button.
- ➢ Press "Settings" button.
- ➢ Click "Update & Security".
- ➢ Choose "For developers" on the left and choose "Developers mode" on the right.

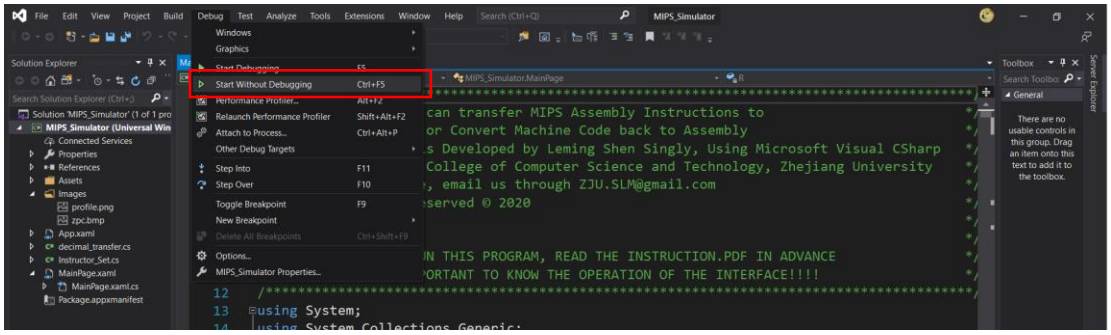




3.2. Build & Run

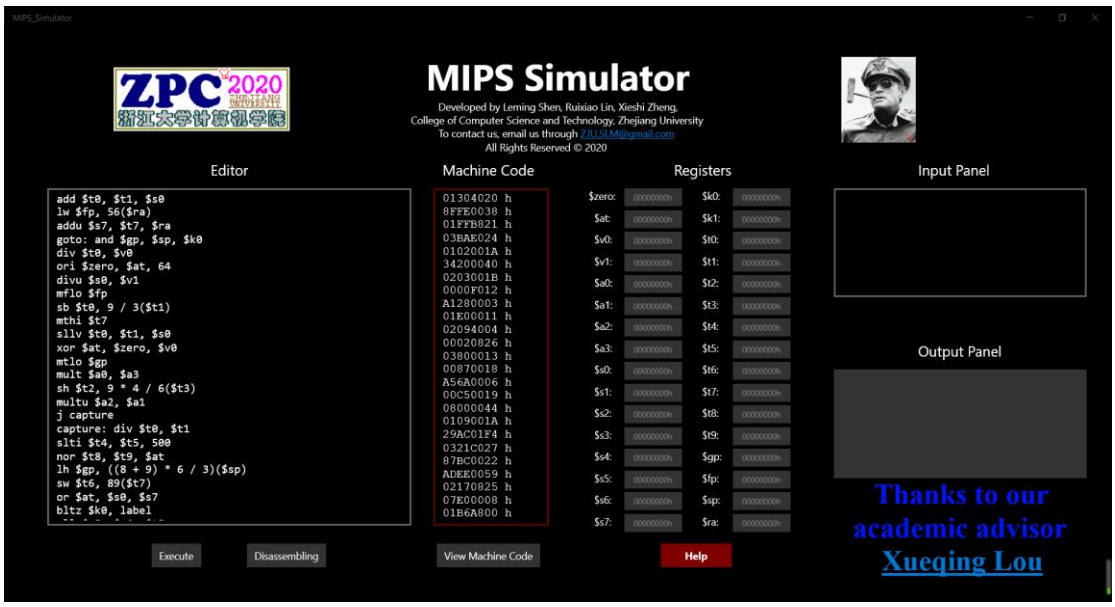
3.2.1. Open Visual Studio 2019 and open the solution file or directly open the "MIPS_Simulator.sln" file through Visual Studio. And you will see as follow:

3.2.2. To run the program, click "Local Machine" on the top or click "Debug" in the menu list and choose "Start without Debugging". After a while, you will see the UWP application.

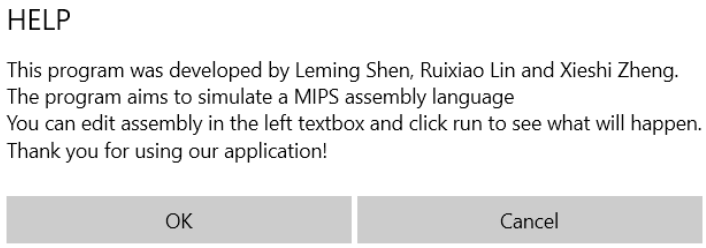


3.3. Interface

3.3.1. After running the program, the interface is attached and its operation will also be attached:



- On the mid-top of the app is the app's introduction. And if you have any question, you can click the e-mail and send an e-mail to me.
- Especially thanks to the academic advisor, Professor Xueqing Lou. You can click his name to view his personal information through website.
- MIPS Assembly source code is written on the left textbox. You can copy the sample code from a txt file named "text.txt" in folder "MIPS_Simulator".
- The corresponding machine code is shown on the middle textbox.
- The button named "Help" helps you better understand the app. After clicking it, a message dialog will be shown:



- When you complete the assembly code, you can click the button named "View Machine Code" to see the corresponding machine code. (Shown in the above figure)
- When you complete the machine code, you can click the button named "Disassemble" to see the corresponding assembly code. (Also shown in the above figure)
- If your input contain fault, the app will send you an error message dialog like follow:

- The "Register" Area shows each register's value every time after executing the code.
- The two panel on the right are respectively input and output panel, in which you can input and watch output result.

3.4. Input & Output

3.4.1. Input Standard (VERY IMPORTANT)

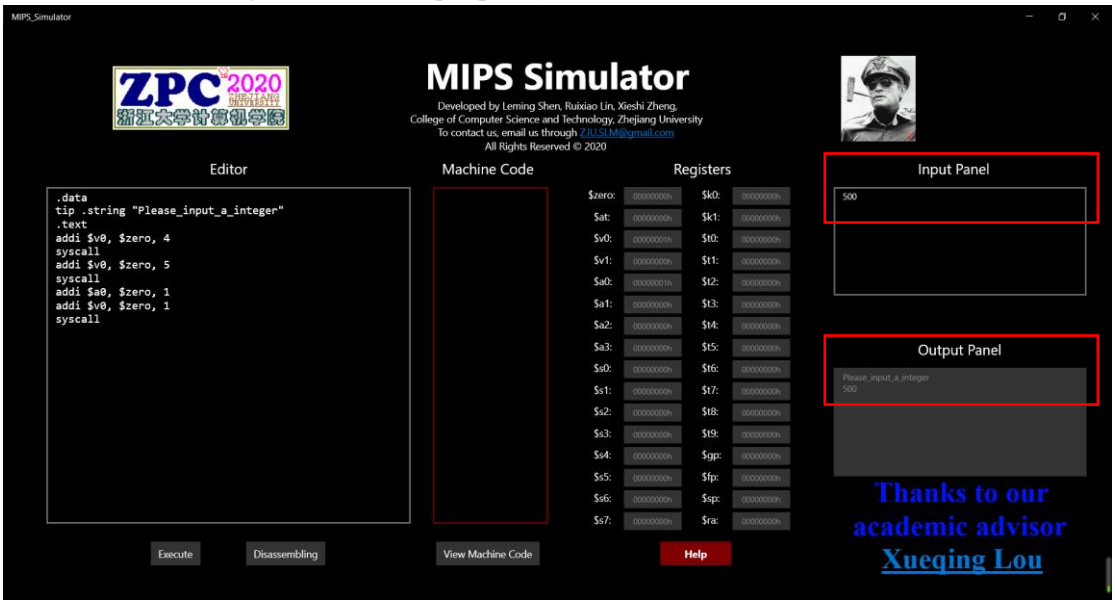
- There should be no space at the begin of any code line.
- The assembly code does NOT support commence, that is char '#' is not allowed.
- The assembly code does NOT support pseudo instruction, for the sake of convenience!!!!!
- Every register's name should be correct or the app will send you an error dialog.
- Every instruction should obey the rule in the PDF document named "ZPC 之 MIPS 指令集".
- When you input an expression instead of immediate number, the expression must be computable and valid. Expressions like "9 / 6" or "2 ^ 6" are not allowed. (The division in C# is allowed only in integers with exact division)
- The translating part only supports 4 types of instructions, they are all listed in chapter

5. Pseudo instruction and formation instruction are also not allowed.

- The executing part only supports 3 types of instructions, except Coprocessor Instruction.
- The first 8 bit of the machine code must be valid hex string.
- If the app tells that you have a "Label not found" error, please check carefully that whether every label is defined or not.

3.4.2. Output

- When you click "View Machine Code" the output machine code will be shown in the textbox named "Machine Code".
- When you click "Disassemble" the output assembly code will be shown in the textbox named "Editor".
- When you click "Execute" the registers' value after execution will be shown in the register area, the output panel will show outputs if necessary. (Don't forget to input necessary values in int input panel)

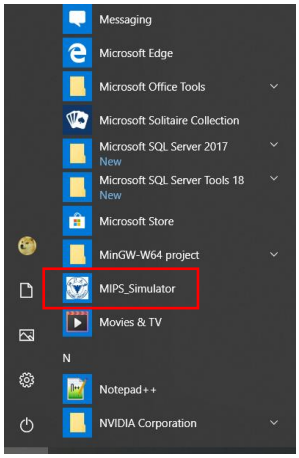


3.5. Execution

There are two method to execute the program.

1. Launch Visual Studio 2019 Profession Version and run it through the Visual Studio Integrated Development Environment. And don't forget to open your computer's developers mode.
2. If you have already built the solution for at least one time, you can directly find the app in your start menu list.

→→→



3.6. Exception Handling

The program can handle these exceptions:

- ✓ Invalid Operation Exception
- ✓ Not Implemented Exception
- ✓ Not Supported Exception
- ✓ Argument Exception
- ✓ Index Out of Range Exception
- ✓ Rank Exception
- ✓ Timeout Exception
- ✓ I/O Exception
- ✓ Array Type Mismatch Exception
- ✓ Null Reference Exception
- ✓ Divide by Zero Exception
- ✓ Invalid Cast Exception
- ✓ Out of Memory Exception
- ✓ Stack Overflow Exception
- ✓ Others (marked as Unknown)

4. Maintenance

- ➢ During which you use the app, if you meet any problem, you can directly contact me through e-mail ZJU.SLM@gmail.com or 3180103654@zju.edu.cn .
- ➢ You can also visit my Facebook website https://www.facebook.com/leming.shen.33, my Twitter website https://twitter.com/ZJU_SLM, my YouTube website https://www.youtube.com/channel/UCv2UyFij9dC5H7sQm_UySGg?view_as=subscriber, my Instagram website https://www.instagram.com/shenleming/.

5. More & About

5.1. About the Software

5.1.1. The supported assembly instructions:

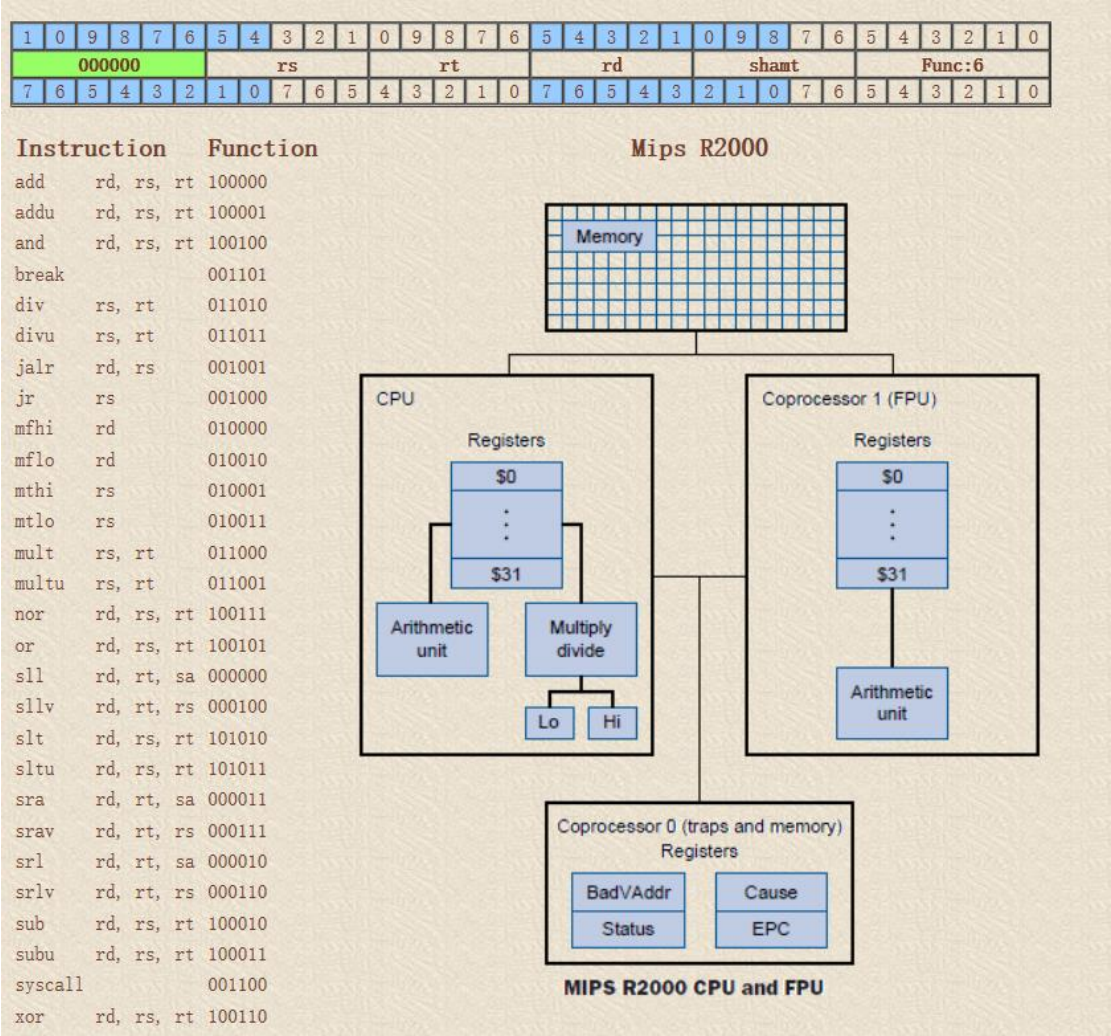
- ➢ R-Type Instructions (Operation Code 000000)

  Main processor instructions that do not require a target address, immediate value, or

branch displacement use an R-Type coding format. This format has fields for specifying op up to three registers and a shift amount.

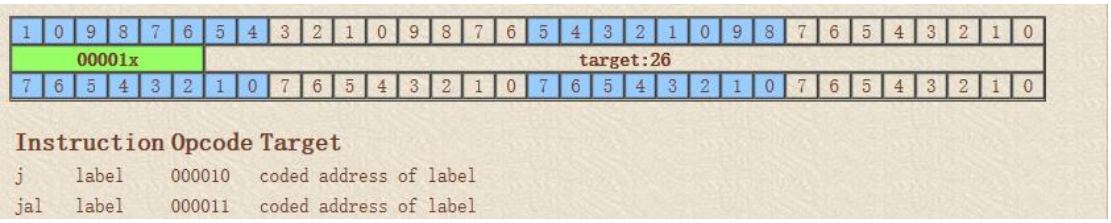
For instructions that do not use all of these fields, the unused fields are coded with all 0 bits. All R-Type instructions use a 000000 operation code. The operation is specified by the function field.

The “shamt” part are all coded with 00000.



- J-Type Instruction (Operation Code 00001x)

The only J-Type instructions are the jump instructions j and jal. These instructions require a 26-bit coded address field to specify the target of the jump. The coded address is formed from the bits at positions 27 to 2 in the binary representation of the address. The bits at positions 1 and 0 are always 0 since instructions are word-aligned.



When a J-Type instruction is executed, a full 32-bit jump target address is formed by concatenating the high order four of the PC (the address of the instruction following the jump), the 26 bits of the target field, and two 0 digits.

- I-Type Instruction (All Operation Codes except 000000, 00001x, and 0100xx)

I-Type instructions have a 16-bit immediate field that codes an immediate operand, a branch target offset, or a displacement for a memory operand. For a branch target offset, the immediate field contains the signed difference between the address of the following instruction and the target label, with the two low order bits dropped. The dropped bits are always 0 since instructions are word-aligned.

For the bgez, bgtz, blez and bltz instructions, the rt field is used as an extension of the operation code field.

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
OP:6						rs:5						rt:5				immediate:16															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Instruction	Opcode	Notes
addi rt, rs, immediate	001000	
addiu rt, rs, immediate	001001	
andi rt, rs, immediate	001100	
beq rs, rt, label	000100	
bgez rs, label	000001	rt = 00001
bgtz rs, label	000111	rt = 00000
blez rs, label	000110	rt = 00000
bltz rs, label	000001	rt = 00000
bne rs, rt, label	000101	
lb rt, immediate(rs)	100000	
lbu rt, immediate(rs)	100100	
lh rt, immediate(rs)	100001	
lhu rt, immediate(rs)	100101	
lui rt, immediate	001111	
lw rt, immediate(rs)	100011	
lwc1 rt, immediate(rs)	110001	
ori rt, rs, immediate	001101	
sb rt, immediate(rs)	101000	
slti rt, rs, immediate	001010	
sltiu rt, rs, immediate	001011	
sh rt, immediate(rs)	101001	
sw rt, immediate(rs)	101011	
swc1 rt, immediate(rs)	111001	
xori rt, rs, immediate	001110	

- Coprocessor Instruction (Operation Code 0100xx)

The only instructions that are described here are the floating point instructions that are common to all processors in the MIPS family. All coprocessor instructions use operation code 0100xx. The last two bits specify the coprocessor number. Thus all floating point instructions use operation code 010001.

The instruction is broken up into fields of the same sizes as in the R-Type instruction format. However, the fields are used in different ways.

Most floating point instructions use the format field to specify a numerical coding format: single precision(.s), double precision(.d), or fixed point(.w). The mfcl and mtcl instructions uses the format field as an extension of the function field.

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0100xx						format					ft					fs						fd					Func:6				
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Instruction	Function	Format
add.s fd, fs, ft	000000	10000
cvt.s.w fd, fs, ft	100000	10100
cvt.w.s fd, fs, ft	100100	10000
div.s fd, fs, ft	000011	10000
mfc1 ft, fs	000000	00000
mov.s fd, fs	000110	10000
mtc1 ft, fs	000000	00100
mul.s fd, fs, ft	000010	10000
sub.s fd, fs, ft	000001	10000

5.1.2. Basic Information

The software “MISP Simulator” is developed singly by Leming Shen, College of Computer Science and Technology, Zhejiang University, major in Software Engineering.

5.2. About the Developer

- ✓ Name: Leming Shen
- ✓ Student ID: 3180103654
- ✓ Phone: 15381145750
- ✓ E-Mail:
  - ZJU.SLM@gmail.com
  - 3180103654@zju.edu.cn
- ✓ Personal CSDN Website: https://me.csdn.net/James_Bond_slm

  Personal GitHub Website: https://github.com/ZJUslm

  Personal LeetCode Website: https://leetcode.com/zju_slm/

  Personal Facebook Website: https://www.facebook.com/leming.shen.33

  Personal Twitter Website: https://twitter.com/ZJU_SLM

  Personal YouTube Website: https://www.youtube.com/channel/UCv2UyFij9dC5H7sQm_UySGg

  Personal Instagram website https://www.instagram.com/shenleming/

Appendix

1. Test Code

   A. Test for assembling and disassembling

```
add $t0, $t1, $s0
lw $fp, 56($ra)
addu $s7, $t7, $ra
goto: and $gp, $sp, $k0
div $t0, $v0
ori $zero, $at, 64
divu $s0, $v1
mflo $fp
sb $t0, 9 / 3($t1)
mthi $t7
sllv $t0, $t1, $s0
xor $at, $zero, $v0
mtlo $gp
mult $a0, $a3
sh $t2, 9 * 4 / 6($t3)
multu $a2, $a1
j capture
capture: div $t0, $t1
slti $t4, $t5, 500
nor $t8, $t9, $at
lh $gp, ((8 + 9) * 6 / 3)($sp)
sw $t6, 89($t7)
or $at, $s0, $s7
bltz $k0, label
sll $s5, $s6, $t5
jalr $s0, $t0
bne $k1, $ra, goto
jr $t2
mfhi $s0
break
syscall
andi $s0, $t0, 102
label: bne $t0, $zero, label
exit: bgez $at, exit
bgtz $k0, capture
blez $a3, cap
sub $s2, $s3, $s4
xori $t8, $t9, (((8 + 9) * 7 + 6) * (3 + 4) * 6 / (7 + 14))
```

```
subu $s5, $s6, $s7
lbu $ra, 3 * 4($sp)
lui $fp, (10 + 3) * 6
add.s $gp, $sp, $t9
mfcl $t0, $s7
mov.s $t7, $s7
slt $gp, $sp, $fp
sltu $ra, $at, $zero
sra $t0, $t1, $t2
srav $t3, $t4, $t5
cap: addi $v1, $a0, 10 % 2
srl $t6, $t7, $t8
srlv $t9, $s0, $s1
jal label
lb $ra, (3 + 6) * 9($t0)
beq $a1, $a2, cap
```

B. Print out Hello World

```
.data
message .string "Hello_World!"
.text
addi $a0, $zero, 0
addi $v0, $zero, 4
syscall
```

C. Test for input

```
.data
tip .string "Please_input_a_integer"
.text
addi $v0, $zero, 4
syscall
addi $v0, $zero, 5
syscall
addi $a0, $zero, 1
addi $v0, $zero, 1
syscall
```

D. Test for multiple variables

```
.data
var1 .int 500
var2 .int 600
.text
```

```
addi $v0, $zero, 1
syscall
addi $a0, $zero, 1
syscall
```

E. Test for simple loop

```
.data
ch .string "*"
.text
addi $t0, $zero, 5
addi $v0, $zero, 4
addi $a0, $zero, 0
loop: addi $t0, $t0, -1
syscall
bne $t0, $zero, loop
```

2. Source Code

A. MainPage.xaml

```
<Page
    x:Class="MIPS_Simulator.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MIPS_Simulator"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <TextBlock x:Name="Title" HorizontalAlignment="Center" Margin="0,30,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="80" Width="366" FontSize="50" FontWeight="Bold"><Run Text="MIPS Simulator"/><LineBreak/><Run/></TextBlock>

        <Button Content="Help" Margin="910,711,0,0" VerticalAlignment="Top" Width="98" Click="help" HorizontalAlignment="Left" RequestedTheme="Default" Background="#7FFF0000" FontWeight="Bold"/>
        <TextBlock HorizontalAlignment="Center" Margin="0,97,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="88" Width="408"><Run Text="          Developed by Leming Shen, Ruixiao Lin, Xieshi Zheng,"/><LineBreak/><Run Text="College of Computer Science and Technology, Zhejiang University"/><LineBreak/><Run Text="          To contact us, email us through "/><Hyperlink NavigateUri="mailto:ZJ
```

```
U.SLM@gmail.com">ZJU.SLM@gmail.com</Hyperlink><LineBreak/><Run Text="
                    All Rights Reserved © 2020"/></TextBlock>
        <TextBlock HorizontalAlignment="Left" Margin="286,180,0,0" Text="Editor" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="20"/>
        <Button x:Name="Execute" Content="Execute" Margin="205,711,0,0" VerticalAlignment="Top" Click="Execute_and_Check"/>
        <TextBlock HorizontalAlignment="Left" Margin="608,180,0,0" Text="Machine Code" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="20"/>
        <TextBlock HorizontalAlignment="Left" Margin="929,180,0,0" Text="Registers" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="20" Width="80"/>
        <TextBox x:Name="zero" HorizontalAlignment="Left" Margin="860,220,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,220,0,0" Text="$zero:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,250,0,0" Text="  $at:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,280,0,0" Text="  $v0:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,310,0,0" Text="  $v1:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,340,0,0" Text="  $a0:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,370,0,0" Text="  $a1:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,400,0,0" Text="  $a2:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,430,0,0" Text="  $a3:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,460,0,0" Text="  $s0:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,490,0,0" Text="  $s1:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,520,0,0" Text="  $s2:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,550,0,0" Text="  $s3:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,580,0,0" Text="  $s4:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,610,0,0" Text="  $s5:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="810,640,0,0" Text="  $s6:" TextWrapping="Wrap" VerticalAlignment="Top"/>
```

```
        <TextBlock HorizontalAlignment="Left" Margin="810,670,0,0" Text="  $s7:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBox x:Name="at" HorizontalAlignment="Left" Margin="860,250,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="v0" HorizontalAlignment="Left" Margin="860,280,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="v1" HorizontalAlignment="Left" Margin="860,310,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="a0" HorizontalAlignment="Left" Margin="860,340,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="a1" HorizontalAlignment="Left" Margin="860,370,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="a2" HorizontalAlignment="Left" Margin="860,400,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="a3" HorizontalAlignment="Left" Margin="860,430,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s0" HorizontalAlignment="Left" Margin="860,460,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s1" HorizontalAlignment="Left" Margin="860,490,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s2" HorizontalAlignment="Left" Margin="860,520,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s3" HorizontalAlignment="Left" Margin="860,550,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s4" HorizontalAlignment="Left" Margin="860,580,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s5" HorizontalAlignment="Left" Margin="860,610,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="s6" HorizontalAlignment="Left" Margin="860,640,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
```

```
        <TextBox x:Name="s7" HorizontalAlignment="Left" Margin="860,670,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="k0" HorizontalAlignment="Left" Margin="1010,220,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,220,0,0" Text="  $k0:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,250,0,0" Text="  $k1:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,280,0,0" Text="  $t0:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,310,0,0" Text="  $t1:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,340,0,0" Text="  $t2:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,370,0,0" Text="  $t3:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,400,0,0" Text="  $t4:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,430,0,0" Text="  $t5:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,460,0,0" Text="  $t6:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,490,0,0" Text="  $t7:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,520,0,0" Text="  $t8:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,550,0,0" Text="  $t9:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,580,0,0" Text="  $gp:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,610,0,0" Text="  $fp:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,640,0,0" Text="  $sp:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="960,670,0,0" Text="  $ra:" TextWrapping="Wrap" VerticalAlignment="Top"/>
        <TextBox x:Name="k1" HorizontalAlignment="Left" Margin="1010,250,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t0" HorizontalAlignment="Left" Margin="1010,280,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
```

```
        <TextBox x:Name="t1" HorizontalAlignment="Left" Margin="1010,310,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t2" HorizontalAlignment="Left" Margin="1010,340,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t3" HorizontalAlignment="Left" Margin="1010,370,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t4" HorizontalAlignment="Left" Margin="1010,400,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t5" HorizontalAlignment="Left" Margin="1010,430,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t6" HorizontalAlignment="Left" Margin="1010,460,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t7" HorizontalAlignment="Left" Margin="1010,490,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t8" HorizontalAlignment="Left" Margin="1010,520,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="t9" HorizontalAlignment="Left" Margin="1010,550,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="gp" HorizontalAlignment="Left" Margin="1010,580,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="fp" HorizontalAlignment="Left" Margin="1010,610,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="sp" HorizontalAlignment="Left" Margin="1010,640,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <TextBox x:Name="ra" HorizontalAlignment="Left" Margin="1010,670,0,0" Text="00000000h" TextWrapping="Wrap" VerticalAlignment="Top" IsEnabled="False" Height="25" FontSize="11" Width="80"/>
        <Button Content="View Machine Code" Margin="600,711,0,0" VerticalAlignment="Top" Click="view_machine_code"/>
        <TextBlock HorizontalAlignment="Left" Margin="1267,430,0,0" Text="Output Panel" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="20" Width="121"/>
```

```
        <TextBlock HorizontalAlignment="Left" Margin="1267,180,0,0" Text="Input Panel" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="20" Width="121"/>
        <TextBox x:Name="output" HorizontalAlignment="Left" Margin="1150,470,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="12" IsEnabled="False" Height="150" Width="350"/>
        <TextBox x:Name="input" HorizontalAlignment="Left" Margin="1150,220,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="12" Height="150" Width="350" AcceptsReturn="True"/>
        <TextBlock HorizontalAlignment="Left" Margin="1177,620,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="146" Width="301" FontSize="40" FontWeight="Bold" FontFamily="Times New Roman" FocusVisualPrimaryBrush="#FF0031FF" Foreground="#FF0015FF"><Run Text="   Thanks to our "/><LineBreak/><Run Text="academic advisor"/><LineBreak/><Run Text="    "></Run><Hyperlink NavigateUri="https://person.zju.edu.cn/hzlou">Xueqing Lou</Hyperlink></TextBlock>
        <Image HorizontalAlignment="Left" Height="88" Margin="152,53,0,0" VerticalAlignment="Top" Width="244" Source="/Images/zpc.bmp"/>
        <Image HorizontalAlignment="Left" Height="118" Margin="1161,38,0,0" VerticalAlignment="Top" Width="103" Source="/Images/profile.png" Stretch="Fill"/>
        <TextBox x:Name="source_code" HorizontalAlignment="Left" Margin="60,220,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" Width="505" Height="466" FontSize="16" FontWeight="Normal" FontFamily="Consolas" AcceptsReturn="True" IsSpellCheckEnabled="False" RequestedTheme="Dark"/>
        <Button Content="Disassembling" Margin="337,711,0,0" VerticalAlignment="Top" Click="Disassemble"/>
        <TextBox x:Name="Machine_Code" HorizontalAlignment="Left" Margin="595,220,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" Height="466" Width="160" FontFamily="Courier New" FontSize="16" BorderBrush="#66FF0000" AcceptsReturn="True"/>

    </Grid>
</Page>
```

B. decimal_transfer.cs

```
/* This class contains all varieties of decimal transfering functions */

using System;


/*-------------------------> main part of class decimal_transfer <-------------------------*/
namespace MIPS_Simulator
{
    class decimal_transfer
    {
```

```
        /* convert an integer to a binary form in certain bits */
        public string int_to_bin_bit(int x, int bit)
        {
            string temp = "", result = "";
            int i = 0;
            long xx = x >= 0 ? x : (long)((long)Math.Pow(2, bit) + (long)x);
            while (i < bit)
            {
                temp += (xx % 2).ToString();
                xx /= 2;
                i++;
            }
            for (int j = temp.Length - 1; j >= 0; j--)
            {
                result += temp[j];
            }

            return result;
        }

        /* convert a binary string into interger and return */
        public int bin_to_int(string bin)
        {
            int result = 0;
            int fact = 1;
            for (int i = bin.Length - 1; i >= 0; i--)
            {
                result += (bin[i] - '0') * fact;
                fact *= 2;
            }

            return result;
        }

        /* convert a binary string into a hex string */
        public string bin_to_hex(string bin)
        {
            string result = "";
            string[] hex = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F" };
            for (int i = 0; i < 8; i++)
            {
                result += hex[bin_to_int(bin.Substring(i * 4, 4))];
            }
```

```

            return result;
        }

        /* convert a hex string into a binary string */
        public string hex_to_bin(string hex)
        {
            string result = "";
            for (int i = 0; i < hex.Length; i++)
            {
                if (hex[i] == '0')
                {
                    result += "0000";
                }
                if (hex[i] == '1')
                {
                    result += "0001";
                }
                if (hex[i] == '2')
                {
                    result += "0010";
                }
                if (hex[i] == '3')
                {
                    result += "0011";
                }
                if (hex[i] == '4')
                {
                    result += "0100";
                }
                if (hex[i] == '5')
                {
                    result += "0101";
                }
                if (hex[i] == '6')
                {
                    result += "0110";
                }
                if (hex[i] == '7')
                {
                    result += "0111";
                }
                if (hex[i] == '8')
                {
```

```
                    result += "1000";
                }
                if (hex[i] == '9')
                {
                    result += "1001";
                }
                if (hex[i] == 'A')
                {
                    result += "1010";
                }
                if (hex[i] == 'B')
                {
                    result += "1011";
                }
                if (hex[i] == 'C')
                {
                    result += "1100";
                }
                if (hex[i] == 'D')
                {
                    result += "1101";
                }
                if (hex[i] == 'E')
                {
                    result += "1110";
                }
                if (hex[i] == 'F')
                {
                    result += "1111";
                }
            }
            return result;
        }

        /* convert an integer into a hex string */
        public string int_to_hex(int x)
        {
            string result = "", temp = "";
            string bin = int_to_bin_bit(x, 32);
            return bin_to_hex(bin);
        }
    }
}
```

C. Instructor_Set.cs

```
using System.Collections.Generic;

/*-------------------------> main part of class Instructor_Set <---------------
-----------*/
namespace MIPS_Simulator
{
    class Instructor_Set
    {
        public string int_to_bin_bit(int x, int bit)
        {
            string temp = "", result = "";
            int i = 0;
            while (i < bit)
            {
                temp += (x % 2).ToString();
                x /= 2;
                i++;
            }
            for (int j = temp.Length - 1; j >= 0; j--)
            {
                result += temp[j];
            }
            return result;
        }

        public int bin_to_int(string bin)
        {
            int result = 0;
            int fact = 1;
            for (int i = bin.Length - 1; i >= 0; i--)
            {
                result += (bin[i] - '0') * fact;
                fact *= 2;
            }

            return result;
        }



        /*-------------------------> Main part of the function set <-----------
---------------*/
        public void add(ref int destiny, int operand1, int operand2)
```

```
        {
            destiny = operand1 + operand2;
            return;
        }

        public void sub(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 - operand2;
            return;
        }

        public void and(ref int destiny, int operand1, int operand2)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(operand2, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += (int.Parse(bit1[i].ToString()) & int.Parse(bit2[i].ToString())).ToString();
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void nor(ref int destiny, int operand1, int operand2)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(operand2, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += (~(int.Parse(bit1[i].ToString()) | int.Parse(bit2[i].ToString()))).ToString();
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void or(ref int destiny, int operand1, int operand2)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(operand2, 32);
            string bit3 = "";
```

```
            for (int i = 0; i < 32; i++)
            {
                bit3 += (int.Parse(bit1[i].ToString()) | int.Parse(bit2[i].ToStr
    ing())).ToString();
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void slt(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 < operand2 ? 0 : 1;
            return;
        }

        public void xor(ref int destiny, int operand1, int operand2)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(operand2, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += bit1[i] == bit2[i] ? "0" : "1";
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void sll(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 << operand2;
            return;
        }

        public void sllv(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 << operand2;
            return;
        }

        public void srl(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 >> operand2;
            return;
```

```
        }

        public void srlv(ref int destiny, int operand1, int operand2)
        {
            destiny = operand1 >> operand2;
            return;
        }

        public void sra(ref int destiny, int operand1, int operand2)
        {
            if (destiny == 0)
            {
                destiny = 0;
            }
            else if (destiny > 0)
            {
                for (int i = 0; i < operand2; i++)
                {
                    operand1 /= 2;
                }
                destiny = operand1;
            }
            else
            {
                for (int i = 0; i < operand2; i++)
                {
                    operand1 /= 2;
                }
                destiny = operand1 - 1;
            }
            return;
        }

        public void srav(ref int destiny, int operand1, int operand2)
        {
            if (destiny == 0)
            {
                destiny = 0;
            }
            else if (destiny > 0)
            {
                for (int i = 0; i < operand2; i++)
                {
                    operand1 /= 2;
```

```
                }
                destiny = operand1;
            }
            else
            {
                for (int i = 0; i < operand2; i++)
                {
                    operand1 /= 2;
                }
                destiny = operand1 - 1;
            }
            return;
        }

        public void div(ref int lo, ref int hi, int operand1, int operand2)
        {
            lo = operand1 / operand2;
            hi = operand1 % operand2;
            return;
        }

        public void divu(ref int lo, ref int hi, int operand1, int operand2)
        {
            lo = operand1 / operand2;
            hi = operand1 % operand2;
            return;
        }

        public void mult(ref int lo, ref int hi, int operand1, int operand2)
        {
            long result = (long)operand2 * (long)operand1;
            hi = (int)(((long)result >> 32) & 0xFFFFFFFF);
            lo = (int)(((long)result) & 0xFFFFFFFF);
            return;
        }

        public void multu(ref int lo, ref int hi, int operand1, int operand2)
        {
            long result = (long)operand2 * (long)operand1;
            hi = (int)(((long)result >> 32) & 0xFFFFFFFF);
            lo = (int)(((long)result) & 0xFFFFFFFF);
            return;
        }

```

```
        public void mfhi(ref int reg, int hi)
        {
            reg = hi;
            return;
        }

        public void mflo(ref int reg, int lo)
        {
            reg = lo;
            return;
        }

        public void mthi(ref int hi, int reg)
        {
            hi = reg;
            return;
        }

        public void mtlo(ref int lo, int reg)
        {
            lo = reg;
            return;
        }

        public void addi(ref int destiny, int operand1, int immeadiate)
        {
            destiny = operand1 + immeadiate;
            return;
        }

        public void andi(ref int destiny, int operand1, int immeadiate)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(immeadiate, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += (int.Parse(bit1[i].ToString()) & int.Parse(bit2[i].ToSt
    ring())).ToString();
            }
            destiny = bin_to_int(bit3);
            return;
        }

```

```
        public void ori(ref int destiny, int operand1, int immeadiate)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(immeadiate, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += (int.Parse(bit1[i].ToString()) | int.Parse(bit2[i].ToSt
ring())).ToString();
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void xori(ref int destiny, int operand1, int immeadiate)
        {
            string bit1 = int_to_bin_bit(operand1, 32);
            string bit2 = int_to_bin_bit(immeadiate, 32);
            string bit3 = "";
            for (int i = 0; i < 32; i++)
            {
                bit3 += bit1[i] == bit2[i] ? "0" : "1";
            }
            destiny = bin_to_int(bit3);
            return;
        }

        public void slti(ref int destiny, int operand1, int immeadiate)
        {
            destiny = operand1 < immeadiate ? 1 : 0;
            return;
        }

        public void lw(ref List<string> Memory, ref int destiny, int offset, in
t register)
        {
            destiny = bin_to_int(Memory[register + offset]);
            return;
        }

        public void lh(ref List<string> Memory, ref int destiny, int offset, in
t register)
        {
            destiny = bin_to_int(Memory[offset + destiny].Substring(16, 16));
```

```
            return;
        }

        public void sw(ref List<string> Memory, int source, int offset, int register)
        {
            Memory[offset + register] = int_to_bin_bit(source, 32);
        }

        public void sh(ref List<string> Memory, int destiny, int offset, int register)
        {
            Memory[offset + register] = int_to_bin_bit(destiny / 65536, 32);
        }

        public void lui(ref int register, int data)
        {
            register = data << 16;
        }
    }
}
```

D. MainPage.xaml.cs

```
/*****************************************************************************************/
/*      This program can transfer MIPS Assembly Instructions to                          */
/*      Machine Code or Convert Machine Code back to Assembly.                           */
/*      It could also manipulate the assembly code and show the result                   */
/*      of registers changing and output in "registers" & "output panel"                 */
/*      The Program is Developed by Leming Shen Singly, Using Microsoft Visual C Sharp   */
/*      Leming Shen, College of Computer Science and Technology, Zhejiang University     */
/*      To contact me, email us through                                                  */
/*      ZJU.SLM@gmail.com or 3180103654@zju.edu.cn                                       */
/*      All Rights Reserved © 2020                                                       */
```

```
/*
          */
/*      WARNING!!!!
          */
/*      BEFORE YOU RUN THIS PROGRAM, READ THE INSTRUCTION.PDF IN ADVANCE
          */
/*      IT'S VERY IMPORTANT TO KNOW THE OPERATION OF THE INTERFACE!!!!
          */
/*****************************************************************************
*********/


/*-------------------------> definition of important library <-----------------
---------*/
using System;
using System.Collections.Generic;
using System.Data;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

// The Blank Page item template is documented at https://go.microsoft.com/fwlink
/?LinkId=402352&clcid=0x409


/*-------------------------> Main part using namespace of project's name <-----
---------------------*/
namespace MIPS_Simulator
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        /*-------------------------> definition of important variables <-------
-------------------*/
        /* The arguement list, an O(n) time index search */
        string[] R = { "add", "addu", "and", "break", "div", "divu", "jalr", "jr
", "mfhi", "mflo", "mthi", "mtlo", "mult", "multu", "nor", "or", "sll", "sllv",
"slt", "sltu", "sra", "srav", "srl", "srlv", "sub", "subu", "syscall", "xor" };
```

```
        string[] R_function = { "100000", "100001", "100100", "001101", "011010", "011011", "001001", "001000", "010000", "010010", "010001", "010011", "011000", "011001", "100111", "100101", "000000", "000100", "101010", "101011", "000011", "000111", "000010", "000110", "100010", "100011", "011100", "100110" };
        string[] I = { "addi", "addiu", "andi", "beq", "bgez", "bgtz", "blez", "bltz", "bne", "lb", "lbu", "lh", "lhu", "lui", "lw", "lwcl", "ori", "sb", "slti", "sltiu", "sh", "sw", "swcl", "xori" };
        string[] I_function = { "001000", "001001", "001100", "000100", "000001", "000111", "000110", "000001", "000101", "100000", "100100", "100001", "100101", "001111", "100011", "110001", "001101", "101000", "001010", "001011", "101001", "101011", "111001", "001110" };
        string[] J = { "j", "jal" };
        string[] J_function = { "000010", "000011" };
        string[] C = { "add.s", "cvt.s.w", "cvt.w.s", "div.s", "mfcl", "mov.s", "mtcl", "mul.s", "sub.s" };
        string[] C_function = { "000000", "100000", "100100", "000011", "000000", "000110", "000000", "000010", "000001" };
        string[] C_Format = { "10000", "10100", "10000", "10000", "00000", "10000", "00100", "10000", "10000" };
        string[] registers = { "$zero", "$at", "$v0", "$v1", "$a0", "$a1", "$a2", "$a3", "$t0", "$t1", "$t2", "$t3", "$t4", "$t5", "$t6", "$t7", "$s0", "$s1", "$s2", "$s3", "$s4", "$s5", "$s6", "$s7", "$t8", "$t9", "$k0", "$k1", "$gp", "$sp", "$fp", "$ra" };
        int reg_lo = 0, reg_hi = 0; /* 32 bit */

        /* all decimal transfer operations are encapsulated into a class named decimal_transfer */
        decimal_transfer deci = new decimal_transfer();

        /* the space for code & memory segment need to be specified */
        List<string> Memory = new List<string>();
        List<string> Code_Segment = new List<string>();

        /* stores declared variables */
        List<List<string>> Variable = new List<List<string>>();

        /* the stack segment */
        Stack<string> stack = new Stack<string>();



        /*--------------------------> Page Initialization <----------------------*/
```

```
        public MainPage()
        {
            this.InitializeComponent();
        }



        /*--------------------------> help button <--------------------------
*/
        /* when press the button "help", a message dialog will come out */
        private void help(object sender, RoutedEventArgs e)
        {
            ContentDialog help = new ContentDialog();
            help.Title = "HELP";
            help.Content = "This program was developed by Leming Shen, Ruixiao L
in and Xieshi Zheng.\nThe program aims to simulate a MIPS assembly language\nYou
 can edit assembly in the left textbox and click run to see what will happen.\nT
hank you for using our application!";
            help.SecondaryButtonText = "Cancel";
            help.PrimaryButtonText = "OK";
            _ = help.ShowAsync();
        }



        /*--------------------------> synchronize the registers' value <-------
------------------*/
        /* transfer each register's value into hex string */
        public void reg_to_text(int[] reg)
        {
            zero.Text = "00000000h";
            at.Text = deci.int_to_hex(reg[1]) + "h";
            v0.Text = deci.int_to_hex(reg[2]) + "h";
            v1.Text = deci.int_to_hex(reg[3]) + "h";
            a0.Text = deci.int_to_hex(reg[4]) + "h";
            a1.Text = deci.int_to_hex(reg[5]) + "h";
            a2.Text = deci.int_to_hex(reg[6]) + "h";
            a3.Text = deci.int_to_hex(reg[7]) + "h";
            t0.Text = deci.int_to_hex(reg[8]) + "h";
            t1.Text = deci.int_to_hex(reg[9]) + "h";
            t2.Text = deci.int_to_hex(reg[10]) + "h";
            t3.Text = deci.int_to_hex(reg[11]) + "h";
            t4.Text = deci.int_to_hex(reg[12]) + "h";
            t5.Text = deci.int_to_hex(reg[13]) + "h";
```

```
            t6.Text = deci.int_to_hex(reg[14]) + "h";
            t7.Text = deci.int_to_hex(reg[15]) + "h";
            s0.Text = deci.int_to_hex(reg[16]) + "h";
            s1.Text = deci.int_to_hex(reg[17]) + "h";
            s2.Text = deci.int_to_hex(reg[18]) + "h";
            s3.Text = deci.int_to_hex(reg[19]) + "h";
            s4.Text = deci.int_to_hex(reg[20]) + "h";
            s5.Text = deci.int_to_hex(reg[21]) + "h";
            s6.Text = deci.int_to_hex(reg[22]) + "h";
            s7.Text = deci.int_to_hex(reg[23]) + "h";
            t8.Text = deci.int_to_hex(reg[24]) + "h";
            t9.Text = deci.int_to_hex(reg[25]) + "h";
            k0.Text = deci.int_to_hex(reg[26]) + "h";
            k1.Text = deci.int_to_hex(reg[27]) + "h";
            gp.Text = deci.int_to_hex(reg[28]) + "h";
            sp.Text = deci.int_to_hex(reg[29]) + "h";
            fp.Text = deci.int_to_hex(reg[30]) + "h";
            ra.Text = deci.int_to_hex(reg[31]) + "h";
        }


        /*-------------------------
-> execute button to manipulate the assembly code <-------------------------
*/
        /* Execute the source code at once */
        private void Execute_and_Check(object sender, RoutedEventArgs e)
        {
            /* Initialize the Memeory, stack and variable segment */
            Memory.Clear();
            Variable.Clear();
            stack.Clear();
            output.Text = "";

            int current_line = 0;

            /* the class Instructor_Set contains function to manipulate instructors */
            Instructor_Set set = new Instructor_Set();

            try
            {
                bool data = false;                                  /* represents whether it comes to vatiable declaration segment */
```

```
                int num_of_null_line = 0;                           /* stores the null line number */
                string function;                                    /* stores the function of each instructor */
                string instructors = source_code.Text;              /* the original source code from input textox */
                string[] instructor_set = instructors.Split('\r');  /* split the whole string into many lines */
                int num_of_code_line = instructor_set.Length;       /* calculate the line number */
                int[] label_line = new int[100];                    /* stroes every label's line number */
                string[] label_name = new string[100];              /* stores evety lael's name corresponding to line number*/
                int index_label = 0;                                /* represents the index of current label */
                /* stores 32 registers value */
                int[] reg = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
                DataTable dt = new DataTable();

                /* Before manipulation, get information of each code's machine code
                 * and they are all stored in List Code_Segment
                 */
                for (int i = 0; i < num_of_code_line; i++)
                {
                    current_line = i + 1;

                    /* when meet the null line, just ignore and continue */
                    if (instructor_set[i] == "")
                    {
                        num_of_null_line++;
                        continue;
                    }

                    /* split each line and get the separate string */
                    string[] temp = instructor_set[i].Split(' ');

                    /* dealing with labels */
                    if (instructor_set[i].IndexOf(':') != -1) /* this means this line contains a label */
                    {
                        function = temp[1];
```

```
                        label_line[index_label] = i - num_of_null_line;
                        label_name[index_label] = temp[0].Substring(0, temp[0].Length - 1);
                        index_label++;
                        int len = temp.Length;

                        if (len > 2)
                        {
                            for (int j = 0; j < len - 2; j++)
                            {
                                temp[j] = temp[j + 2];
                            }
                            temp[len - 1] = "";
                            temp[len - 2] = "";
                        }
                        else
                        {
                            for (int j = 0; i < len - 1; j++)
                            {
                                temp[j] = temp[j + 1];
                            }
                            temp[len - 1] = "";
                        }
                    }
                    else
                    {
                        function = temp[0];     /* the function then is the first string */
                        int len = temp.Length;

                        if (len > 1)
                        {
                            for (int j = 0; j < len - 1; j++)
                            {
                                temp[j] = temp[j + 1];
                            }
                            temp[len - 1] = "";
                        }
                    }

                    /* function is the main part of the code,
                     * representing instructor type */
                    function = function.ToLower();

```

```
                    /* check which type of instructions the function is */
                    int index_R = Array.IndexOf(R, function);
                    int index_I = Array.IndexOf(I, function);
                    int index_J = Array.IndexOf(J, function);
                    int index_C = Array.IndexOf(C, function);

                    string current_machine_code = "";

                    /* get three or less registers' name */
                    instructor_set[i] = String.Join("", temp);
                    temp = instructor_set[i].Split(',');

                    if (index_R != -1) /* R type */
                    {
                        current_machine_code += "000000";

                        /* deal with defferent kinds of R type instructions
                         * noted that different R type instructions' registers' code place are various.
                         */
                        if (function == "add" || function == "addu" || function == "and" || function == "nor" || function == "or" || function == "slt" || function == "sltu" || function == "sub" || function == "subu" || function == "xor")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else if (function == "sllv" || function == "sll" || function == "sra" || function == "srav" || function == "srl" || function == "srlv")

                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else if (function == "div" || function == "divu" || function == "mult" || function == "multu")
```

```
                        {
                            if ((function == "div" || function == "divu") && temp[1] == "$zero")
                            {
                                MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                                _ = msg.ShowAsync();
                                return;
                            }
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";
                        }
                        else if (function == "jalr")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else if (function == "jr" || function == "mthi" || function == "mtlo")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += "0000000000";
                        }
                        else if (function == "mfhi" || function == "mflo")
                        {
                            current_machine_code += "0000000000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else
                        {
                            current_machine_code += "000000000000000";
                        }

                        current_machine_code += "00000" + R_function[index_R];
```

```
                    }
                    else if (index_I != -1) /* similar to above */
                    {
                        int flag = 1; /* stands for whether the label is found later */
                        current_machine_code += I_function[index_I];

                        if (function == "addi" || function == "addiu" || function == "andi" || function == "xori" || function == "ori" || function == "slti" || function == "sltiu")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[2], "false").ToString()), 16);
                        }
                        else if (function == "beq" || function == "bne")
                        {
                            flag = 1;
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);

                            int jump_index = Array.IndexOf(label_name, temp[2]);
                            /* calculate the subtraction between current line's next address and the target jump address */
                            if (jump_index != -1)
                            {
                                current_machine_code += deci.int_to_bin_bit(label_line[jump_index] - (i + 1 - num_of_null_line), 16);
                            }
                            else
                            {
                                int temp_num = num_of_null_line;

                                for (int j = i + 1; j < num_of_code_line; j++)

                                {
                                    if (instructor_set[j] == "")
                                    {
```

```
                                    temp_num++;
                                    continue;
                                }
                                if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[2])
                                {
                                    flag = 0;
                                    current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                    break;
                                }
                            }
                            if (flag == 1)
                            {
                                throw new InvalidOperationException("Label not found!"); ;
                            }
                        }
                    }
                    else if (function == "bgez")
                    {
                        flag = 1;
                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        current_machine_code += "00001";

                        int jump_index = Array.IndexOf(label_name, temp[1]);

                        if (jump_index != -1)
                        {
                            current_machine_code += deci.int_to_bin_bit(jump_index - (i + 1 - num_of_null_line), 16);
                        }
                        else
                        {
                            int temp_num = num_of_null_line;

                            for (int j = i + 1; j < num_of_code_line; j++)
                            {
                                if (instructor_set[j] == "")
                                {
```

```
                                    temp_num++;
                                    continue;
                                }
                                if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[1])
                                {
                                    flag = 0;
                                    current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                    break;
                                }
                            }
                            if (flag == 1)
                            {
                                throw new InvalidOperationException("Label not found!"); ;
                            }
                        }
                    }
                    else if (function == "bgtz" || function == "blez" || function == "bltz")
                    {
                        flag = 1;
                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                        current_machine_code += "00000";

                        int jump_index = Array.IndexOf(label_name, temp[1]);

                        if (jump_index != -1)
                        {
                            current_machine_code += deci.int_to_bin_bit(jump_index - (i + 1 - num_of_null_line), 16);
                        }
                        else
                        {
                            int temp_num = num_of_null_line;

                            for (int j = i + 1; j < num_of_code_line; j++)

                            {
                                if (instructor_set[j] == "")
```

```
                                    {
                                        temp_num++;
                                        continue;
                                    }
                                    if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[1])
                                    {
                                        flag = 0;
                                        current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                        break;
                                    }
                                }
                                if (flag == 1)
                                {
                                    throw new InvalidOperationException("Label not found!"); ;
                                }
                            }
                        }
                        else if (function == "lb" || function == "lbu" || function == "lh" || function == "lhu" || function == "lw" || function == "lwcl" || function == "sb" || function == "sh" || function == "sw" || function == "swcl")
                        {
                            int j = 0;

                            for (j = temp[1].Length - 1; j >= 0; j--)
                            {
                                if (temp[1][j] == '(')
                                {
                                    break;
                                }
                            }

                            string rs = temp[1].Substring(j + 1, 3);

                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, rs), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString()), 16);
                        }
```

```
                        else
                        {
                            current_machine_code += "00000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[1], "false").ToString()), 16);
                        }
                    }
                    else if (index_J != -1) /* the J type instruction's last 26 bits' machine code the the target address direact */
                    {
                        current_machine_code += J_function[index_J];
                        int jump_index = Array.IndexOf(label_name, temp[0]);

                        if (jump_index != -1)
                        {
                            current_machine_code += deci.int_to_bin_bit(label_line[jump_index] * 4, 26);
                        }
                        else
                        {
                            int temp_num = num_of_null_line;

                            for (int j = i + 1; j < num_of_code_line; j++)
                            {
                                if (instructor_set[j] == "")
                                {
                                    temp_num++;
                                    continue;
                                }
                                if (instructor_set[j].IndexOf(':') != -1)
                                {
                                    current_machine_code += deci.int_to_bin_bit((j - temp_num) * 4, 26);
                                    break;
                                }
                            }
                        }
                    }
                    else if (index_C != -1) /* other corporation instruction */
                    {
```

```
                        current_machine_code += "010001";
                        current_machine_code += C_Format[index_C];

                        if (function == "add.s" || function == "cvt.s.w" || function == "cvt.w.s" || function == "div.s" || function == "mul.s" || function == "sub.s")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else if (function == "mfcl" || function == "mtcl")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";
                        }
                        else
                        {
                            current_machine_code += "00000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        current_machine_code += C_function[index_C];
                    }

                    Code_Segment.Add(current_machine_code);
                }

                /* Here comes the main part of execution */
                for (int i = 0; i < num_of_code_line; i++)
                {
                    instructor_set = instructors.Split('\r');
                    current_line = i + 1;

                    /* when meet the null line, just ignore and continue */
                    if (instructor_set[i] == "")
```

```
                    {
                        num_of_null_line++;
                        continue;
                    }
                    /* split each line and get the separate string */
                    string[] temp = instructor_set[i].Split(' ');

                    /* deal with labels the same as above */
                    if (instructor_set[i].IndexOf(':') != -1) /* this means this line contains a label */
                    {
                        function = temp[1];
                        label_line[index_label] = i - num_of_null_line;
                        label_name[index_label] = temp[0].Substring(0, temp[0].Length - 1);
                        index_label++;
                        int len = temp.Length;

                        if (len > 2)
                        {
                            for (int j = 0; j < len - 2; j++)
                            {
                                temp[j] = temp[j + 2];
                            }
                            temp[len - 1] = "";
                            temp[len - 2] = "";
                        }
                        else
                        {
                            for (int j = 0; i < len - 1; j++)
                            {
                                temp[j] = temp[j + 1];
                            }
                            temp[len - 1] = "";
                        }
                    }
                    else
                    {
                        function = temp[0];     /* the function then is the first string */
                        int len = temp.Length;

                        if (len > 1)
                        {
```

```
                        for (int j = 0; j < len - 1; j++)
                        {
                            temp[j] = temp[j + 1];
                        }
                        temp[len - 1] = "";
                    }
                }

                function = function.ToLower();

                /* check which type of instructions the function is */
                int index_R = Array.IndexOf(R, function);
                int index_I = Array.IndexOf(I, function);
                int index_J = Array.IndexOf(J, function);
                int index_C = Array.IndexOf(C, function);

                /* get three or less registers' name */
                instructor_set[i] = String.Join("", temp);
                temp = instructor_set[i].Split(',');

                /* differ the function and do different work */
                if (function == "add")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.add(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "sub" || function == "subu")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.sub(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "and" || function == "addu")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);
```

```
                    set.and(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "nor")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.nor(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "or")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.or(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "slt" || function == "sltu")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.slt(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "xor")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.slt(ref reg[destiny], reg[operand1], reg[operand2]);
                }

                else if (function == "sll")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
```

```
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = int.Parse(temp[2]);

                    set.sll(ref reg[destiny], reg[operand1], operand2);
                }
                else if (function == "sllv")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.sll(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "sra")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = int.Parse(temp[2]);

                    set.sra(ref reg[destiny], reg[operand1], operand2);
                }
                else if (function == "srav")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);

                    set.srav(ref reg[destiny], reg[operand1], reg[operand2]);
                }
                else if (function == "srl")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = int.Parse(temp[2]);

                    set.srl(ref reg[destiny], reg[operand1], operand2);
                }
                else if (function == "srlv")
                {
                    int destiny = Array.IndexOf(registers, temp[0]);
                    int operand1 = Array.IndexOf(registers, temp[1]);
                    int operand2 = Array.IndexOf(registers, temp[2]);
```

```
                        set.srlv(ref reg[destiny], reg[operand1], reg[operand2]);
                    }
                    else if (function == "mult")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);
                        int operand2 = Array.IndexOf(registers, temp[1]);

                        set.mult(ref reg_lo, ref reg_hi, reg[operand1], reg[operand2]);
                    }
                    else if (function == "multu")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);
                        int operand2 = Array.IndexOf(registers, temp[1]);

                        set.multu(ref reg_lo, ref reg_hi, reg[operand1], reg[operand2]);
                    }
                    else if (function == "div")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);
                        int operand2 = Array.IndexOf(registers, temp[1]);

                        /* dealing with divided by zero exception */
                        if (reg[operand2] == 0)
                        {
                            MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                            _ = msg.ShowAsync();
                            return;
                        }
                        else
                        {
                            set.div(ref reg_lo, ref reg_hi, operand1, operand2);
                        }
                    }
                    else if (function == "divu")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);
                        int operand2 = Array.IndexOf(registers, temp[1]);
```

```
                        if (reg[operand2] == 0)
                        {
                            MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                            _ = msg.ShowAsync();
                            return;
                        }
                        else
                        {
                            set.divu(ref reg_lo, ref reg_hi, operand1, operand2);
                        }
                    }
                    else if (function == "jalr")
                    {
                        int address = Array.IndexOf(registers, temp[0]);
                        int next = Array.IndexOf(registers, temp[1]);

                        /* when the jump address is too large, throw an exception */
                        if (reg[address] > Code_Segment.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        reg[next] = current_line + 1;
                        reg[31] = current_line + 1;
                        i = reg[address] - 1;
                        continue;
                    }
                    else if (function == "jr")
                    {
                        int address = Array.IndexOf(registers, temp[0]);

                        if (reg[address] > Code_Segment.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        i = reg[address] - 1;
                        continue;
                    }
```

```
                    else if (function == "mfhi")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);

                        set.mfhi(ref reg[operand1], reg_hi);
                    }
                    else if (function == "mflo")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);

                        set.mflo(ref reg[operand1], reg_lo);
                    }
                    else if (function == "mthi")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);

                        set.mthi(ref reg_hi, reg[operand1]);
                    }
                    else if (function == "mtlo")
                    {
                        int operand1 = Array.IndexOf(registers, temp[0]);

                        set.mtlo(ref reg_lo, reg[operand1]);
                    }
                    else if (function == "break")
                    {
                        MessageDialog msg = new MessageDialog("The program break at line " + current_line.ToString());
                        _ = msg.ShowAsync();

                        return;
                    }
                    else if (function == "syscall")
                    {
                        if (reg[2] == 1 || reg[2] == 2 || reg[2] == 3 || reg[2] == 4 || reg[2] == 11)
                        {
                            output.Text += Variable[reg[4]][2] + "\n";
                        }
                        if (reg[2] == 5 || reg[2] == 6 || reg[2] == 7 || reg[2] == 8 || reg[2] == 12)
                        {
                            List<string> tempo = new List<string>();

```

```
                            tempo.Add("unknown_name");
                            tempo.Add("unknown_type");
                            tempo.Add(input.Text);

                            Variable.Add(tempo);
                        }
                        if (reg[2] == 10 || reg[2] == 17)
                        {
                            MessageDialog msg = new MessageDialog("The program exit at line " + current_line.ToString());
                            _ = msg.ShowAsync();
                            return;
                        }
                    }
                    else if (function == "addi" || function == "addiu")
                    {
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int operand1 = Array.IndexOf(registers, temp[1]);
                        int immeadiate = int.Parse(dt.Compute(temp[2], "false").ToString());

                        set.addi(ref reg[destiny], reg[operand1], immeadiate);

                    }
                    else if (function == "andi")
                    {
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int operand1 = Array.IndexOf(registers, temp[1]);
                        int immeadiate = int.Parse(dt.Compute(temp[2], "false").ToString());

                        set.andi(ref reg[destiny], reg[operand1], immeadiate);

                    }
                    else if (function == "ori")
                    {
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int operand1 = Array.IndexOf(registers, temp[1]);
                        int immeadiate = int.Parse(dt.Compute(temp[2], "false").ToString());

                        set.ori(ref reg[destiny], reg[operand1], immeadiate);
                    }
                    else if (function == "xori")
```

```
                    {
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int operand1 = Array.IndexOf(registers, temp[1]);
                        int immeadiate = int.Parse(dt.Compute(temp[2], "false")
    .ToString());

                        set.xori(ref reg[destiny], reg[operand1], immeadiate);

                    }
                    else if (function == "slti" || function == "sltiu")
                    {
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int operand1 = Array.IndexOf(registers, temp[1]);
                        int immeadiate = int.Parse(dt.Compute(temp[2], "false")
    .ToString());

                        set.slti(ref reg[destiny], reg[operand1], immeadiate);

                    }
                    else if (function == "lw" || function == "lwcl")
                    {
                        /* deal with empty memory but want to load from memory
    */
                        if (Memory.Count == 0)
                        {
                            MessageDialog msg = new MessageDialog("The Memory i
    s empty! Wrong at line " + current_line.ToString());
                            _ = msg.ShowAsync();
                            return;
                        }

                        int j = 0;

                        for (j = temp[1].Length - 1; j >= 0; j--)
                        {
                            if (temp[1][j] == '(')
                            {
                                break;
                            }
                        }

                        string rs = temp[1].Substring(j + 1, 3);
                        int register = Array.IndexOf(registers, rs);
                        int destiny = Array.IndexOf(registers, temp[0]);
```

```
                        int offset = int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString());

                        if (offset + reg[register] > Memory.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        set.lw(ref Memory, ref reg[destiny], offset, reg[register]);
                    }
                    else if (function == "lh" || function == "lhu")
                    {
                        if (Memory.Count == 0)
                        {
                            MessageDialog msg = new MessageDialog("The Memory is empty! Wrong at line " + current_line.ToString());
                            _ = msg.ShowAsync();
                            return;
                        }

                        int j = 0;

                        for (j = temp[1].Length - 1; j >= 0; j--)
                        {
                            if (temp[1][j] == '(')
                            {
                                break;
                            }
                        }

                        string rs = temp[1].Substring(j + 1, 3);
                        int register = Array.IndexOf(registers, rs);
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int offset = int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString());

                        if (offset + reg[register] > Memory.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        set.lh(ref Memory, ref reg[destiny], offset, reg[register]);
```

```
                    }
                    else if (function == "sh")
                    {
                        int j = 0;

                        for (j = temp[1].Length - 1; j >= 0; j--)
                        {
                            if (temp[1][j] == '(')
                            {
                                break;
                            }
                        }

                        string rs = temp[1].Substring(j + 1, 3);
                        int register = Array.IndexOf(registers, rs);
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int offset = int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString());

                        if (offset + reg[register] > Memory.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        set.sh(ref Memory, reg[destiny], offset, reg[register]);
                    }
                    else if (function == "sw" || function == "swcl")
                    {
                        int j = 0;

                        for (j = temp[1].Length - 1; j >= 0; j--)
                        {
                            if (temp[1][j] == '(')
                            {
                                break;
                            }
                        }

                        string rs = temp[1].Substring(j + 1, 3);
                        int register = Array.IndexOf(registers, rs);
                        int destiny = Array.IndexOf(registers, temp[0]);
                        int offset = int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString());
```

```
                        if (offset + reg[register] > Memory.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        set.sw(ref Memory, destiny, offset, register);
                    }
                    else if (function == "lui")
                    {
                        int register = Array.IndexOf(registers, temp[0]);
                        int dataa = int.Parse(dt.Compute(temp[1], "false").ToSt
    ring());

                        set.lui(ref reg[register], dataa);
                    }
                    else if (function == "bne")
                    {
                        int num1 = Array.IndexOf(registers, temp[0]);
                        int num2 = Array.IndexOf(registers, temp[1]);
                        int jump_index = Array.IndexOf(label_name, temp[2]);

                        if (label_line[jump_index] > Code_Segment.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        if (jump_index != -1)
                        {
                            if (reg[num1] != reg[num2])
                            {
                                i = label_line[jump_index] - 1;
                                continue;
                            }
                        }
                        else    /* the jump index is invalid */
                        {
                            throw new InvalidOperationException();
                        }
                    }
                    else if (function == "beq")
                    {
                        int num1 = Array.IndexOf(registers, temp[0]);
                        int num2 = Array.IndexOf(registers, temp[1]);
```

```
                        int jump_index = Array.IndexOf(label_name, temp[2]);

                        if (label_line[jump_index] > Code_Segment.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        if (jump_index != -1)
                        {
                            if (reg[num1] == reg[num2])
                            {
                                i = label_line[jump_index] - 1;
                                continue;
                            }
                        }
                        else
                        {
                            throw new InvalidOperationException();
                        }
                    }
                        else if (function == "j")
                        {
                            int jump_index = Array.IndexOf(label_name, temp[0]);

                            if (label_line[jump_index] > Code_Segment.Count)
                            {
                                throw new IndexOutOfRangeException();
                            }

                            if (jump_index != -1)
                            {
                                i = label_line[jump_index] - 1;
                                continue;
                            }
                            else
                            {
                                throw new InvalidOperationException();
                            }
                        }
                        else if (function == "jal")
                        {
                            reg[31] = i + 1;
```

```
                        int jump_index = Array.IndexOf(label_name, temp[0]);

                        if (label_line[jump_index] > Code_Segment.Count)
                        {
                            throw new IndexOutOfRangeException();
                        }

                        if (jump_index != -1)
                        {
                            i = label_line[jump_index] - 1;
                            continue;
                        }
                        else
                        {
                            throw new InvalidOperationException();
                        }
                    }
                    else if (function == ".data")
                    {
                        data = true;
                        continue;
                    }
                    else if (function == ".text")
                    {
                        data = false;
                        continue;
                    }
                    else if (data)
                    {
                        /* the first element stores the name, second stores the type, third stores value */
                        List<string> tempo = new List<string>();
                        tempo.Add(function);

                        if (temp[0].IndexOf("string") != -1)
                        {
                            tempo.Add(temp[0].Substring(0, 7));
                            tempo.Add(temp[0].Substring(8, temp[0].Length - 9));
                        }
                        if (temp[0].IndexOf("int") != -1)
                        {
                            tempo.Add(temp[0].Substring(0, 4));
```

```
                                tempo.Add(temp[0].Substring(4, temp[0].Length - 4));
                            }
                            if (temp[0].IndexOf("float") != -1)
                            {
                                tempo.Add(temp[0].Substring(0, 6));
                                tempo.Add(temp[0].Substring(6, temp[0].Length - 6));
                            }
                            if (temp[0].IndexOf("double") != -1)
                            {
                                tempo.Add(temp[0].Substring(0, 7));
                                tempo.Add(temp[0].Substring(7, temp[0].Length - 7));
                            }
                            if (temp[0].IndexOf("char") != -1)
                            {
                                tempo.Add(temp[0].Substring(0, 5));
                                tempo.Add(temp[0].Substring(6, temp[0].Length - 7));
                            }

                            Variable.Add(tempo);
                        }
                        else
                        {
                            MessageDialog msg = new MessageDialog("Unsupported operation!!");
                            _ = msg.ShowAsync();
                            throw new NotSupportedException();
                        }

                        /* synchronize the registers' value */
                        reg_to_text(reg);
                    }
                }
                /* Here comes the exception dealing part, it can catch the corresponding fault and send messages */
                catch (InvalidOperationException err)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: " + err.Message + "\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
```

```
                    //System.DBNull.Value
                }
                catch (NotImplementedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NotImplementedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NotSupportedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID:  NotSupportedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArgumentException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArgumentException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (IndexOutOfRangeException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IndexOutOfRangeException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (RankException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: RankException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (TimeoutException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: TimeoutException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
```

```
                }
                catch (System.IO.IOException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IOException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArrayTypeMismatchException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArrayTypeMismatchException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NullReferenceException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NullReferenceException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (DivideByZeroException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (InvalidCastException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: InvalidCastException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (OutOfMemoryException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: OutOfMemoryException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (StackOverflowException)
```

```
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: StackOverflowException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (Exception)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: Unknown\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
            }


            /*-------------------------
            -> translate assembly code to machine code <--------------------------*/
            /* when press the button "view machine code",
             * the machine code will be shown in the "machine code" textbox,
             * it will also detect source code grammer and formation error,
             * and will send alert to tell which line occur error
             */
            private void view_machine_code(object sender, RoutedEventArgs e)
            {
                Memory.Clear();
                int current_line = 0;                                   /* stands for current translating line in case for exception */

                try
                {
                    Machine_Code.Text = "";
                    int num_of_null_line = 0;                           /* stores the null line number */
                    string function;                                    /* stores the function of each instructor */
                    string instructors = source_code.Text;              /* the original source code from input textox */
                    string[] instructor_set = instructors.Split('\r');  /* split the whole string into many lines */
                    int num_of_code_line = instructor_set.Length;       /* calculate the line number */
```

```
                    int[] label_line = new int[100];                    /* stores every label's line number */
                    string[] label_name = new string[100];              /* stores evety lael's name corresponding to line number*/
                    int index_label = 0;                                /* represents the index of current label */

                    for (int i = 0; i < num_of_code_line; i++)
                    {
                        current_line = i + 1;

                        /* when meet the null line, just ignore and continue */

                        if (instructor_set[i] == "")
                        {
                            num_of_null_line++;
                            continue;
                        }

                        /* split each line and get the separate string */
                        string[] temp = instructor_set[i].Split(' ');

                        /* dealing with labels */
                        if (instructor_set[i].IndexOf(':') != -1) /* this means this line contains a label */
                        {
                            function = temp[1];
                            label_line[index_label] = i - num_of_null_line;
                            label_name[index_label] = temp[0].Substring(0, temp[0].Length - 1);
                            index_label++;
                            int len = temp.Length;

                            if (len > 2)
                            {
                                for (int j = 0; j < len - 2; j++)
                                {
                                    temp[j] = temp[j + 2];
                                }
                                temp[len - 1] = "";
                                temp[len - 2] = "";
                            }
                            else
                            {
```

```
                                for (int j = 0; i < len - 1; j++)
                                {
                                    temp[j] = temp[j + 1];
                                }
                                temp[len - 1] = "";
                            }
                        }
                        else
                        {
                            function = temp[0];     /* the function then is the first string */
                            int len = temp.Length;

                            if (len > 1)
                            {
                                for (int j = 0; j < len - 1; j++)
                                {
                                    temp[j] = temp[j + 1];
                                }
                                temp[len - 1] = "";
                            }
                        }

                        function = function.ToLower();

                        /* check which type of instructions the function is */

                        int index_R = Array.IndexOf(R, function);
                        int index_I = Array.IndexOf(I, function);
                        int index_J = Array.IndexOf(J, function);
                        int index_C = Array.IndexOf(C, function);

                        string current_machine_code = "";

                        /* get three or less registers' name */
                        instructor_set[i] = String.Join("", temp);
                        temp = instructor_set[i].Split(',');

                        if (index_R != -1) /* R type */
                        {
                            current_machine_code += "000000";

                            /* deal with defferent kinds of R type instructions
```

```
                                 * noted that different R type instructions' registers' code place are various.
                                 */
                                if (function == "add" || function == "addu" || function == "and" || function == "nor" || function == "or" || function == "slt" || function == "sltu" || function == "sub" || function == "subu" || function == "xor")
                                {
                                    current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                    current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                                    current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                                }
                                else if (function == "sllv" || function == "sll" || function == "sra" || function == "srav" || function == "srl" || function == "srlv")
                                {
                                    if (function == "sllv" || function == "srav")
                                    {
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                                    }
                                    else
                                    {
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                        current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                                    }
                                }
                                else if (function == "div" || function == "divu" || function == "mult" || function == "multu")
                                {
                                    if ((function == "div" || function == "divu") && temp[1] == "$zero")
                                    {
```

```
                                MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                                _ = msg.ShowAsync();
                                return;
                            }
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";
                        }
                        else if (function == "jalr")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else if (function == "jr" || function == "mthi" || function == "mtlo")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += "0000000000";
                        }
                        else if (function == "mfhi" || function == "mflo")
                        {
                            current_machine_code += "0000000000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                        }
                        else
                        {
                            current_machine_code += "000000000000000";
                        }

                        current_machine_code += "00000" + R_function[index_R];

                    }
                    else if (index_I != -1) /* similar to above */
```

```
                    {
                        DataTable dt = new DataTable();
                        int flag = 1; /* stands for whether the label is found later */
                        current_machine_code += I_function[index_I];

                        if (function == "addi" || function == "addiu" || function == "andi" || function == "xori" || function == "ori" || function == "slti" || function == "sltiu")
                        {
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[2], "false").ToString()), 16);
                        }
                        else if (function == "beq" || function == "bne")
                        {
                            flag = 1;
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);

                            int jump_index = Array.IndexOf(label_name, temp[2]);

                            /* calculate the subtraction between current line's next address and the target jump address */
                            if (jump_index != -1)
                            {
                                current_machine_code += deci.int_to_bin_bit(label_line[jump_index] - (i + 1 - num_of_null_line), 16);
                            }
                            else
                            {
                                int temp_num = num_of_null_line;

                                for (int j = i + 1; j < num_of_code_line; j++)
                                {
                                    if (instructor_set[j] == "")
                                    {
```

```
                                        temp_num++;
                                        continue;
                                    }
                                    if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[2])
                                    {
                                        flag = 0;
                                        current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                        break;
                                    }
                                }
                                if (flag == 1)
                                {
                                    throw new InvalidOperationException("Label not found!"); ;
                                }
                            }
                        }
                        else if (function == "bgez")
                        {
                            flag = 1;
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += "00001";

                            int jump_index = Array.IndexOf(label_name, temp[1]);

                            if (jump_index != -1)
                            {
                                current_machine_code += deci.int_to_bin_bit(jump_index - (i + 1 - num_of_null_line), 16);
                            }
                            else
                            {
                                int temp_num = num_of_null_line;

                                for (int j = i + 1; j < num_of_code_line; j++)
                                {
                                    if (instructor_set[j] == "")
                                    {
```

```
                                        temp_num++;
                                        continue;
                                    }
                                    if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[1])
                                    {
                                        flag = 0;
                                        current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                        break;
                                    }
                                }
                                if (flag == 1)
                                {
                                    throw new InvalidOperationException("Label not found!"); ;
                                }
                            }
                        }
                        else if (function == "bgtz" || function == "blez" || function == "bltz")
                        {
                            flag = 1;
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                            current_machine_code += "00000";

                            int jump_index = Array.IndexOf(label_name, temp[1]);

                            if (jump_index != -1)
                            {
                                current_machine_code += deci.int_to_bin_bit(jump_index - (i + 1 - num_of_null_line), 16);
                            }
                            else
                            {
                                int temp_num = num_of_null_line;

                                for (int j = i + 1; j < num_of_code_line; j++)
                                {
                                    if (instructor_set[j] == "")
```

```
                                        {
                                            temp_num++;
                                            continue;
                                        }
                                        if (instructor_set[j].IndexOf(':') != -1 && instructor_set[j].Substring(0, instructor_set[j].IndexOf(':')) == temp[1])
                                        {
                                            flag = 0;
                                            current_machine_code += deci.int_to_bin_bit(j - temp_num - (i + 1 - num_of_null_line), 16);
                                            break;
                                        }
                                    }
                                    if (flag == 1)
                                    {
                                        throw new InvalidOperationException("Label not found!"); ;
                                    }
                                }
                            }
                            else if (function == "lb" || function == "lbu" || function == "lh" || function == "lhu" || function == "lw" || function == "lwcl" || function == "sb" || function == "sh" || function == "sw" || function == "swcl")
                            {
                                int j = 0;

                                for (j = temp[1].Length - 1; j >= 0; j--)
                                {
                                    if (temp[1][j] == '(')
                                    {
                                        break;
                                    }
                                }

                                string rs = temp[1].Substring(j + 1, 3);

                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, rs), 5);
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                                current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[1].Substring(0, j), "false").ToString()), 16);
```

```
                        }
                        else
                        {
                            current_machine_code += "00000";
                            current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            current_machine_code += deci.int_to_bin_bit(int.Parse(dt.Compute(temp[1], "false").ToString()), 16);
                        }
                    }
                    else if (index_J != -1) /* the J type instruction's last 26 bits' machine code the the target address direact */
                    {
                        current_machine_code += J_function[index_J];
                        int jump_index = Array.IndexOf(label_name, temp[0]);

                        if (jump_index != -1)
                        {
                            current_machine_code += deci.int_to_bin_bit(label_line[jump_index] * 4, 26);
                        }
                        else
                        {
                            int temp_num = num_of_null_line;

                            for (int j = i + 1; j < num_of_code_line; j++)

                            {
                                if (instructor_set[j] == "")
                                {
                                    temp_num++;
                                    continue;
                                }
                                if (instructor_set[j].IndexOf(':') != -1)
                                {
                                    current_machine_code += deci.int_to_bin_bit((j - temp_num) * 4, 26);
                                    break;
                                }
                            }
                        }
                    }
```

```
                        else if (index_C != -1) /* other corporation instruction */
                        {
                            current_machine_code += "010001";
                            current_machine_code += C_Format[index_C];

                            if (function == "add.s" || function == "cvt.s.w" || function == "cvt.w.s" || function == "div.s" || function == "mul.s" || function == "sub.s")
                            {
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[2]), 5);
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            }
                            else if (function == "mfcl" || function == "mtcl")

                            {
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                current_machine_code += "00000";
                            }
                            else
                            {
                                current_machine_code += "00000";
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[1]), 5);
                                current_machine_code += deci.int_to_bin_bit(Array.IndexOf(registers, temp[0]), 5);
                            }

                            current_machine_code += C_function[index_C];
                        }

                        Machine_Code.Text += deci.bin_to_hex(current_machine_code) + " h\n";
                        Memory.Add(current_machine_code);
                    }
                }
```

```
                /* Here comes the exception dealing part, it can catch the corresponding fault and send messages */
                catch (InvalidOperationException err)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: " + err.Message + "\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();

                    //System.DBNull.Value
                }
                catch (NotImplementedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NotImplementedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NotSupportedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID:  NotSupportedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArgumentException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArgumentException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (IndexOutOfRangeException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IndexOutOfRangeException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (RankException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: RankException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
```

```
                    _ = msg.ShowAsync();
                }
                catch (TimeoutException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: TimeoutException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (System.IO.IOException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IOException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArrayTypeMismatchException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArrayTypeMismatchException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NullReferenceException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NullReferenceException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (DivideByZeroException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (InvalidCastException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: InvalidCastException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
```

```
                catch (OutOfMemoryException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: OutOfMemoryException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (StackOverflowException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: StackOverflowException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (Exception)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: Unknown\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
            }


            /*-------------------------
-> translate machine code back to assembly code <--------------------------*/
            /* when press button "desassemble", the source code textbox will show the corresponding code
             * One more thing needed to pay attention to is that the branch instructions' label will be
             * translated into constant number instead of a label string
             */
            private void Disassemble(object sender, RoutedEventArgs e)
            {
                source_code.Text = "";
                int current_line = 0;

                try
                {
                    string instructors = Machine_Code.Text.ToUpper();   /* uniform the formation of machine code */
                    string[] instruction_set = instructors.Split('\r');
                    int num_of_code_line = instruction_set.Length;
```

```
                    for (int i = 0; i < num_of_code_line; i++)
                    {
                        if (instruction_set[i] == "")
                        {
                            continue;
                        }

                        current_line = i + 1;
                        /* get the binary formation of macine code for the convenience of later recognition */
                        string current_machine_code = deci.hex_to_bin(instruction_set[i].Substring(0, 8));

                        /* the first 6 bit mark the instruction's type */
                        string first_6bit = current_machine_code.Substring(0, 6);
                        string current_assembly_code = "";

                        if (first_6bit == "000000")  // R type
                        {
                            string function = current_machine_code.Substring(26, 6);
                            int index_R = Array.IndexOf(R_function, function);

                            string operation = R[index_R];

                            current_assembly_code += R[index_R] + " ";

                            /* the disasemble process is just the reversion of assemble */
                            if (operation == "add" || operation == "addu" || operation == "and" || operation == "nor" || operation == "or" || operation == "slt" || operation == "sltu" || operation == "sub" || operation == "subu" || operation == "xor")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                            }
```

```
                        else if (operation == "sllv" || operation == "sll" || operation == "sra" || operation == "srav" || operation == "srl" || operation == "srlv")
                        {
                            if (operation == "sllv" || operation == "srav")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))];
                            }
                            else
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(6, 5)).ToString();
                            }
                        }
                        else if (operation == "div" || operation == "divu" || operation == "mult" || operation == "multu")
                        {
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ", ";

                            if ((operation == "div" || operation == "divu") && registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] == "$zero")
                            {
                                MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                                _ = msg.ShowAsync();
                                return;
                            }

                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
```

```
                        else if (operation == "jalr")
                        {
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))];
                        }
                        else if (operation == "jr" || operation == "mthi" || operation == "mtlo")
                        {
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))];
                        }
                        else if (operation == "mfhi" || operation == "mflo")
                        {
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))];
                        }
                        else
                        {
                            current_machine_code += "\n";
                        }
                    }
                    else if (first_6bit == "000010" || first_6bit == "000011") // J type
                    {
                        if (first_6bit == "000010")
                        {
                            current_assembly_code += "j ";
                        }
                        else
                        {
                            current_assembly_code += "jal ";
                        }

                        current_assembly_code += deci.bin_to_int(current_machine_code.Substring(6, 26));
                    }
                    else if (first_6bit == "010001")    /* Coprocessor type, determined by both function and format code */
                    {
                        string function = current_machine_code.Substring(26, 6);
```

```
                        string format = current_machine_code.Substring(6, 5);

                        if (function == "000000" && format == "10000")
                        {
                            current_assembly_code += "add.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                        if (function == "100000" && format == "10100")
                        {
                            current_assembly_code += "cvt.s.w ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                        if (function == "100100" && format == "10000")
                        {
                            current_assembly_code += "cvt.w.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                        if (function == "000011" && format == "10000")
                        {
                            current_assembly_code += "div.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                        if (function == "000000" && format == "00000")
```

```
                        {
                            current_assembly_code += "mfcl ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))];
                        }
                        if (function == "000110" && format == "10000")
                        {
                            current_assembly_code += "mov.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))];
                        }
                        if (function == "000000" && format == "00100")
                        {
                            current_assembly_code += "mtcl ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))];
                        }
                        if (function == "000010" && format == "10000")
                        {
                            current_assembly_code += "mul.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                        if (function == "000001" && format == "10000")
                        {
                            current_assembly_code += "sub.s ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(21, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(16, 5))] + ", ";
                            current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))];
                        }
                    }
```

```
                        else // I type
                        {
                            int index_I = Array.IndexOf(I_function, first_6bit);
                            current_assembly_code += I[index_I] + " ";
                            string function = I[index_I];

                            if (function == "addi" || function == "addiu" || function == "andi" || function == "xori" || function == "ori" || function == "slti" || function == "sltiu")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(16, 16));
                            }
                            else if (function == "beq" || function == "bne")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ", ";
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(16, 16));
                            }
                            else if (function == "bgez" || function == "bgtz" || function == "blez" || function == "bltz")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(16, 16));
                            }
                            else if (function == "lb" || function == "lbu" || function == "lh" || function == "lhu" || function == "lw" || function == "lwcl" || function == "sb" || function == "sh" || function == "sw" || function == "swcl")
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(16, 16)) + "(";
```

```
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(6, 5))] + ")";
                            }
                            else
                            {
                                current_assembly_code += registers[deci.bin_to_int(current_machine_code.Substring(11, 5))] + ", ";
                                current_assembly_code += deci.bin_to_int(current_machine_code.Substring(16, 16));
                            }
                        }

                        source_code.Text += current_assembly_code + "\n";
                    }
                }
                /* Exception dealing procession */
                catch (InvalidOperationException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: InvalidOperationException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();

                    //System.DBNull.Value
                }
                catch (NotImplementedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NotImplementedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NotSupportedException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID:  NotSupportedException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArgumentException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArgumentException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
```

```
                    _ = msg.ShowAsync();
                }
                catch (IndexOutOfRangeException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IndexOutOfRangeException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (RankException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: RankException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (TimeoutException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: TimeoutException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (System.IO.IOException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: IOException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (ArrayTypeMismatchException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: ArrayTypeMismatchException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (NullReferenceException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: NullReferenceException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
```

```
                catch (DivideByZeroException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: DivideByZeroException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (InvalidCastException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: InvalidCastException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (OutOfMemoryException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: OutOfMemoryException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (StackOverflowException)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: StackOverflowException\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
                catch (Exception)
                {
                    MessageDialog msg = new MessageDialog("Wrong in MIPS assembly code!\n Wrong ID: Unknown\n" + "Wrong at line " + current_line.ToString() + "\nPlease check carefully!");
                    _ = msg.ShowAsync();
                }
            }
        }
    }
```