|  |  |
| --- | --- |
| Faculty of Computer & Information Sciences  Ain Shams University  Subject: CHW-362Computer Architecture & Org.  Instructors: Dr. Heba Khaled & Dr. Karim Emara  Year: 3rd year undergraduate  Academic year: 2nd term 2019-2020 |  |

**Practical Project**

**MIPS Pipelined Emulator – Version ( 2)**

**تحذير هام: علي الطالب عدم كتابة اسمه أو كتابة اي شيء يدل علي شخصيته**

**Task2 Report:**

You must write a report for the implemented emulator describing the following:

1. Brief description of your implementation and how the code is organized.

* **Instruction class contain the main data of instruction.**

public class Instruction\_Line

{

public string op\_code;

public string reg\_source;

public string reg\_ttt\_source2;

public string reg\_destenation;

public string shift\_amount;

public string funct;

public string id;

public string instr11\_15;

public string instr16\_20;

public string address\_I\_type16;

public string ALU\_result;

public string instruction\_line\_;

public Instruction\_Line(string instruction\_line, string instruction\_id)

{

instruction\_line\_ = instruction\_line;

op\_code = "";

reg\_source = "";

reg\_ttt\_source2 = "";

reg\_destenation = "";

shift\_amount = "";

funct = "";

instr11\_15 = "";

address\_I\_type16 = "";

ALU\_result = "";

int bit\_position = 0;

id = instruction\_id;

for (int i = instruction\_line.Length - 1; i > instruction\_line.Length - 16; i--)

{

address\_I\_type16 = instruction\_line[i] + address\_I\_type16;

}

for (int i = instruction\_line.Length - 12; i > instruction\_line.Length - 17; i--)

{

instr11\_15 = instruction\_line[i] + instr11\_15;

}

for (int i = bit\_position; i < bit\_position + 6; i++)

{

op\_code += instruction\_line[i];

}

bit\_position += 6;

for (int i = bit\_position; i < bit\_position + 5; i++)

{

reg\_source += instruction\_line[i];

}

bit\_position += 5;

for (int i = bit\_position; i < bit\_position + 5; i++)

{

reg\_ttt\_source2 += instruction\_line[i];

}

bit\_position += 5;

for (int i = bit\_position; i < bit\_position + 5; i++)

{

reg\_destenation += instruction\_line[i];

}

bit\_position += 5;

for (int i = bit\_position; i < bit\_position + 5; i++)

{

shift\_amount += instruction\_line[i];

}

bit\_position += 5;

for (int i = bit\_position; i < bit\_position + 6; i++)

{

funct += instruction\_line[i];

}

bit\_position += 6;

instr16\_20 = reg\_ttt\_source2;

}

}

* **The main data structure in the code id queue of instruction.**

Queue<Instruction\_Line>[] process\_pipeline = new Queue<Instruction\_Line>[6];

* **MIPS Register**

int[] Registers\_\_MIPS;

* **Initialize component (when load form)**

public Form1()

{

InitializeComponent();

//initialize Component "instruction lines"

initialize\_component();

}

* **Global variables**

String iiiInstruction\_line111, iiiInstruction\_line222, iiiInstruction\_line333, iiiInstruction\_line444;

String iiiInstruction\_line555, iiiInstruction\_line666, iiiInstruction\_line777, iiiInstruction\_line888, iiiInstruction\_line999;

String iiiInstruction\_line000;

string[] instruction\_line\_arr1;

string[] instruction\_line\_arr2;

string[] instruction\_line\_arr3;

string[] instruction\_line\_arr4;

string[] instruction\_line\_arr5;

string[] instruction\_line\_arr6;

string[] instruction\_line\_arr7;

string[] instruction\_line\_arr8;

string[] instruction\_line\_arr9;

string program\_counter\_code = "1000";

string read\_data111 = "";

string read\_data222 = "";

string read\_register111 = "";

string read\_registe222 = "";

string write\_data = "";

string write\_register = "";

int number\_of\_cycle = 0;

string sign\_extention = "";

string instruction\_line\_ = "";

string instr\_00\_t0\_15 = "";

string instr\_11\_t0\_15 = "";

string instr\_16\_t0\_20 = "";

string ALU\_Output = "";

string ALU\_mux\_input0 = "";

string reg\_dist\_MUX\_Output = "";

string Address\_mem\_WR = ""; //23

string read\_data25 = ""; //25

string IFetch\_IDecode = "";

string IDecode\_IExecute = "";

string Memory\_write\_back = "";

string key = "";

string value = "";

* **Function initialize\_componemt() to initialize Register\_\_MIPS array and the instructions.**

void initialize\_component()

{

Registers\_\_MIPS = new int[32];

iiiInstruction\_line111 = "1000: 00000000 11000000 00101000 00100101";

iiiInstruction\_line222 = "1004: 00000000 01100100 00010000 00100000";

iiiInstruction\_line333 = "1008: 00000001 00000110 00111000 00100010";

iiiInstruction\_line444 = "1012: 10101111 11000101 00000000 00000010";

iiiInstruction\_line555 = "1016: 00000000 10100000 00110000 00100100";

iiiInstruction\_line666 = "0000: 00000000 00000000 00000000 00000000";

iiiInstruction\_line777 = "0000: 00000000 00000000 00000000 00000000";

iiiInstruction\_line888 = "0000: 00000000 00000000 00000000 00000000";

iiiInstruction\_line999 = "0000: 00000000 00000000 00000000 00000000";

iiiInstruction\_line000 = "0000: 00000000 00000000 00000000 00000000";

Split\_instructions();

Set\_Instruction\_in\_class();

Draw\_Instruction\_text();

}

* **Function Draw\_instructions\_text()**

void Draw\_Instruction\_text() //for start form only

{

string Instruction\_text = "1000: 00000000 11000000 00101000 00100101@1004: 00000000 01100100 00010000 00100000@1008: 00000001 00000110 00111000 00100010@1012: 10101111 11000101 00000000 00000010@1016: 00000000 10100000 00110000 00100100@";

Instruction\_text = Instruction\_text.Replace("@", System.Environment.NewLine);

txt\_user\_code.Text = Instruction\_text;

}

* **Function Split\_instructions()**

void Split\_instructions()

{

instruction\_line\_arr1 = iiiInstruction\_line111.Split(' ');

instruction\_line\_arr1[0] =instruction\_line\_arr1[0].Remove(instruction\_line\_arr1[0].Length - 1); //

instruction\_line\_arr2 = iiiInstruction\_line222.Split(' ');

instruction\_line\_arr2[0] = instruction\_line\_arr2[0].Remove(instruction\_line\_arr2[0].Length - 1); //

instruction\_line\_arr3 = iiiInstruction\_line333.Split(' ');

instruction\_line\_arr3[0] = instruction\_line\_arr3[0].Remove(instruction\_line\_arr3[0].Length - 1); //

instruction\_line\_arr4 = iiiInstruction\_line444.Split(' ');

instruction\_line\_arr4[0] = instruction\_line\_arr4[0].Remove(instruction\_line\_arr4[0].Length - 1); //

instruction\_line\_arr5 = iiiInstruction\_line555.Split(' ');

instruction\_line\_arr5[0] = instruction\_line\_arr5[0].Remove(instruction\_line\_arr5[0].Length - 1); //

instruction\_line\_arr6 = iiiInstruction\_line666.Split(' ');

instruction\_line\_arr6[0] = instruction\_line\_arr6[0].Remove(instruction\_line\_arr6[0].Length - 1); //

instruction\_line\_arr7 = iiiInstruction\_line777.Split(' ');

instruction\_line\_arr7[0] = instruction\_line\_arr7[0].Remove(instruction\_line\_arr7[0].Length - 1); //

instruction\_line\_arr8 = iiiInstruction\_line888.Split(' ');

instruction\_line\_arr8[0] = instruction\_line\_arr8[0].Remove(instruction\_line\_arr8[0].Length - 1); //

instruction\_line\_arr9 = iiiInstruction\_line999.Split(' ');

instruction\_line\_arr9[0] = instruction\_line\_arr9[0].Remove(instruction\_line\_arr9[0].Length - 1); //

}

* **Functions Set\_instruction\_in\_class()**

void Set\_Instruction\_in\_class()

{

string instruction\_line\_str1 = "";

string instruction\_line\_str2 = "";

string instruction\_line\_str3 = "";

string instruction\_line\_str4 = "";

string instruction\_line\_str5 = "";

Instruction\_Line[] Instr\_line = new Instruction\_Line[5];

for (int i = 1; i < 5; i++)

{

instruction\_line\_str1 += instruction\_line\_arr1[i];

instruction\_line\_str2 += instruction\_line\_arr2[i];

instruction\_line\_str3 += instruction\_line\_arr3[i];

instruction\_line\_str4 += instruction\_line\_arr4[i];

instruction\_line\_str5 += instruction\_line\_arr5[i];

}

Instr\_line[0] = new Instruction\_Line(instruction\_line\_str1,instruction\_line\_arr1[0]);

Instr\_line[1] = new Instruction\_Line(instruction\_line\_str2, instruction\_line\_arr2[0]);

Instr\_line[2] = new Instruction\_Line(instruction\_line\_str3, instruction\_line\_arr3[0]);

Instr\_line[3] = new Instruction\_Line(instruction\_line\_str4, instruction\_line\_arr4[0]);

Instr\_line[4] = new Instruction\_Line(instruction\_line\_str5, instruction\_line\_arr5[0]);

process\_pipeline[5] = new Queue<Instruction\_Line>();

for (int i = 0; i < 5; i++)

{

process\_pipeline[i] = new Queue<Instruction\_Line>();

process\_pipeline[0].Enqueue(Instr\_line[i]);

}

for (int i = 0; i < 32; i++)

{

if (i == 0)

{

Registers\_\_MIPS[i] = i;

}

else

Registers\_\_MIPS[i] = i + 100;

}

}

* **initialize\_button()**

private void btn\_initialize\_Click(object sender, EventArgs e)

{

dgv\_mips\_register.Rows.Clear();

Restart();

Set\_instructions\_txt();

for (int i = 0; i < Registers\_\_MIPS.Count(); i++)

{

if (i == 0)

dgv\_mips\_register.Rows.Add("$" + i.ToString(), 0);

else

dgv\_mips\_register.Rows.Add("$" + i.ToString(), Registers\_\_MIPS[i]);

}

}

* **Functions Set\_instruction\_in\_class()**

private void btn\_run\_cycle\_Click(object sender, EventArgs e)

{

for (int i = 4; i >= 0; i--)

{

if (process\_pipeline[i].Count == 0)

{

if (i == 1 && instr\_11\_t0\_15 != "")

{

instr\_00\_t0\_15 = "";

instr\_11\_t0\_15 = "";

instr\_16\_t0\_20 = "";

}

if (i == 2 && number\_of\_cycle == 7)

{

Reset\_varibles();

}

if (i == 1)

{

read\_data111 = "";

read\_data222 = "";

read\_register111 = "";

read\_registe222 = "";

}

continue;

}

int next = i + 1;

Instruction\_Line current\_instruction = process\_pipeline[i].Dequeue();

if (i == 0)

instruction\_line\_ = Fetch\_pipeline(current\_instruction);

if (i == 1)

{

current\_instruction = Decode\_pipeline(current\_instruction);

instr\_11\_t0\_15 = Convert.ToInt32(current\_instruction.instr11\_15, 2).ToString();

instr\_00\_t0\_15 = Convert.ToInt32(current\_instruction.address\_I\_type16, 2).ToString();

instr\_16\_t0\_20 = Convert.ToInt32(current\_instruction.instr16\_20, 2).ToString();

}

if (i == 2)

{

current\_instruction = Execute\_pipeline(current\_instruction);

ALU\_Output = current\_instruction.ALU\_result;

ALU\_mux\_input0 = read\_data222;

reg\_dist\_MUX\_Output = instr\_11\_t0\_15;

}

if (next == 5)

{

write\_data = current\_instruction.ALU\_result;

Address\_mem\_WR = current\_instruction.ALU\_result;

if (current\_instruction.instr11\_15 != "")

write\_register = Convert.ToInt32(current\_instruction.instr11\_15, 2).ToString();

}

else

process\_pipeline[next].Enqueue(current\_instruction);

}

dgv\_pipline.Rows.Clear();

dgv\_pipline.Rows.Add("pc\_counter", program\_counter\_code);

dgv\_pipline.Rows.Add("Current\_instruction", instruction\_line\_);

dgv\_pipline.Rows.Add("read\_Reg1", read\_register111);

dgv\_pipline.Rows.Add("read\_Reg2", read\_registe222);

dgv\_pipline.Rows.Add("read\_Data1", read\_data111);

dgv\_pipline.Rows.Add("read\_Data2", read\_data222);

dgv\_pipline.Rows.Add("instr from 11:15 bits", instr\_11\_t0\_15);

dgv\_pipline.Rows.Add("instr from 00:15 bits", instr\_00\_t0\_15);

dgv\_pipline.Rows.Add("instr from 16:20 bits", instr\_16\_t0\_20);

dgv\_pipline.Rows.Add("ALU Output", ALU\_Output);

dgv\_pipline.Rows.Add("ALU mux input 0", ALU\_mux\_input0);

dgv\_pipline.Rows.Add("reg dist MUX Output", reg\_dist\_MUX\_Output);

dgv\_pipline.Rows.Add("write data", write\_data);

dgv\_pipline.Rows.Add("write register", write\_register);

if (number\_of\_cycle == 7)

{

dgv\_memory\_data.Rows.Add(key, value);

}

number\_of\_cycle++;

MessageBox.Show("the Cycle Number " + number\_of\_cycle.ToString());

}

* **fetch()**

public string Fetch\_pipeline(Instruction\_Line instruction)

{

int program\_counter\_code\_int = int.Parse(program\_counter\_code) + 4;

program\_counter\_code = program\_counter\_code\_int.ToString();

string current\_instructions = instruction.instruction\_line\_;

return current\_instructions;

}

public Instruction\_Line Decode\_pipeline(Instruction\_Line instruction)

{

string[] function\_code = { "100101", "100000", "100010", "101011", "100100" };

//or --> [0]

if (instruction.funct == function\_code[0])

{

int rs = Convert.ToInt32(instruction.reg\_source, 2);

int rt = Convert.ToInt32(instruction.reg\_ttt\_source2, 2);

instruction.reg\_destenation = (rs | rt).ToString();

read\_register111 = rs.ToString();

read\_registe222 = rt.ToString();

read\_data111 = Registers\_\_MIPS[rs].ToString(); //rs

read\_data222 = Registers\_\_MIPS[rt].ToString(); //rt

}

//add -->[1]

else if (instruction.funct == function\_code[1])

{

int rs = Convert.ToInt32(instruction.reg\_source, 2);

int rt = Convert.ToInt32(instruction.reg\_ttt\_source2, 2);

instruction.reg\_destenation = (rs + rt).ToString();

read\_register111 = rs.ToString();

read\_registe222 = rt.ToString();

read\_data111 = Registers\_\_MIPS[rs].ToString();

read\_data222 = Registers\_\_MIPS[rt].ToString();

}

//sub -->[2]

else if (instruction.funct == function\_code[2])

{

int rs = Convert.ToInt32(instruction.reg\_source, 2);

int rt = Convert.ToInt32(instruction.reg\_ttt\_source2, 2);

instruction.reg\_destenation = (rs - rt).ToString();

read\_register111 = rs.ToString();

read\_registe222 = rt.ToString();

read\_data111 = Registers\_\_MIPS[rs].ToString();

read\_data222 = Registers\_\_MIPS[rt].ToString();

}

//store -->[3]

else if (instruction.op\_code == function\_code[3])

{

int rs = Convert.ToInt32(instruction.reg\_source, 2);

int rt = Convert.ToInt32(instruction.reg\_ttt\_source2, 2);

read\_register111 = rs.ToString();

read\_registe222 = rt.ToString();

read\_data111 = Registers\_\_MIPS[rs].ToString();

read\_data222 = Registers\_\_MIPS[rt].ToString();

sign\_extention = (Convert.ToInt32(instruction.address\_I\_type16, 2)).ToString();

}

//and -->[4]

else if (instruction.funct == function\_code[4])

{

int rs = Convert.ToInt32(instruction.reg\_source, 2);

int rt = Convert.ToInt32(instruction.reg\_ttt\_source2, 2);

instruction.reg\_destenation = (rs & rt).ToString();

read\_register111 = rs.ToString();

read\_registe222 = rt.ToString();

read\_data111 = Registers\_\_MIPS[rs].ToString();

read\_data222 = Registers\_\_MIPS[rt].ToString();

}

else

instruction.instr11\_15 = "";

return instruction;

}

public Instruction\_Line Execute\_pipeline(Instruction\_Line instruction)

{

//or

if (instruction.funct == "100101")

{

int data1 = int.Parse(read\_data111);

int data2 = int.Parse(read\_data222);

instruction.ALU\_result = (data1 | data2).ToString();

}

//add

else if (instruction.funct == "100000")

{

int data1 = int.Parse(read\_data111);

int data2 = int.Parse(read\_data222);

instruction.ALU\_result = (data1 + data2).ToString();

}

//sub

else if (instruction.funct == "100010")

{

int data1 = int.Parse(read\_data111);

int data2 = int.Parse(read\_data222);

instruction.ALU\_result = (data1 - data2).ToString();

}

else if (instruction.op\_code == "101011")

{

int data1 = int.Parse(read\_data111);

instruction.ALU\_result = (int.Parse(sign\_extention) + data1).ToString();

key = instruction.ALU\_result;

value = Registers\_\_MIPS[6].ToString();

}

else if (instruction.funct == "100100")

{

int data1 = int.Parse(read\_data111);

int data2 = int.Parse(read\_data222);

instruction.ALU\_result = (data1 & data2).ToString();

}

return instruction;

}

1. High-level pseudo-code or flowchart for “Run 1 cycle” function.

Fetch\_pipeline()

return current\_instructions\_str

Decode\_pipeline(Instruction\_Line instruction)

function\_code = { "100101", "100000", "100010", "101011", "100100" };

if instruction.funct = function\_code of or instructions :

calculte read\_data1

calculte read\_data2

else if instruction.funct = function\_code of add instructions :

calculte read\_data1

calculte read\_data2

else if instruction.funct = function\_code of sub instructions :

calculte read\_data1

calculte read\_data2

else if instruction.funct = function\_code of store instructions :

calculte read\_data1

calculte read\_data2

calculate sign\_extention

else if instruction.funct = function\_code of and instructions :

calculte read\_data1

calculte read\_data2

update all changes in the smae object of instructions and return it

Run one cycle()

for 4 down to 1:

if pipline[i].count = 0:

if i = 1 and bits from 11 to 15 != null:

reset bits

if i = 2:

reset component

continue;

deque pipeline in current\_instructions

if i = 0:

instruction\_str = fitch\_pipline()

if i = 1:

current\_instructions = Decode\_pipeline()

if i = 2:

current\_instructions = Execute\_pipeline()

if next = 5 :

apply memory\_datapipline

clear datagridview

display current cycle

if number\_of\_cycle = 7

diplay (key, value)

number\_of\_cycle ++

Execute\_pipeline(Instruction\_Line instruction)

function\_code = { "100101", "100000", "100010", "101011", "100100" };

if instruction.funct = function\_code of or instructions :

ALU\_result = data1 | data2

else if instruction.funct = function\_code of add instructions :

ALU\_result = data1 + data2

else if instruction.funct = function\_code of sub instructions :

ALU\_result = data1 - data2

else if instruction.funct = function\_code of store instructions :

key = instruction.ALU\_result;

value = Registers\_\_MIPS[6]

else if instruction.funct = function\_code of and instructions :

ALU\_result = data1 & data2

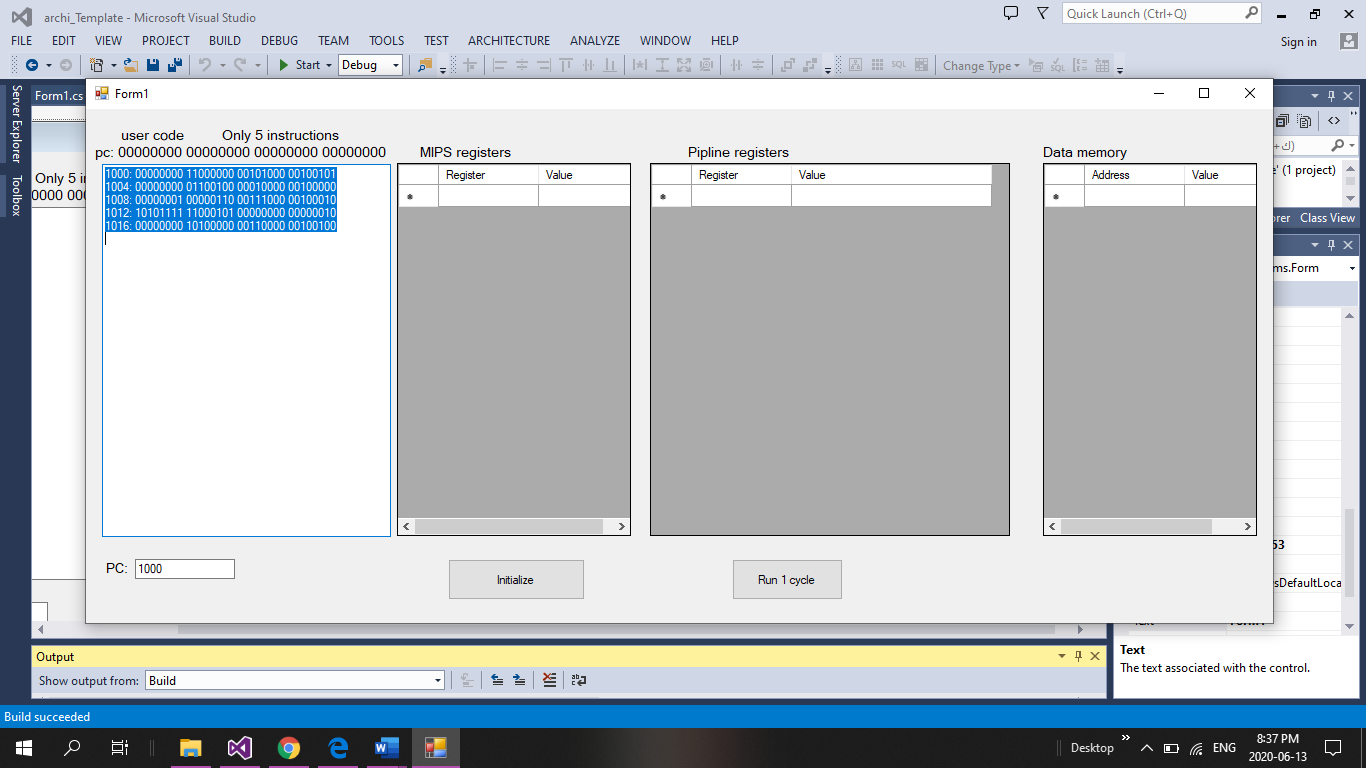
update all changes in the same object of instructions and return it

1. Any assumptions not mentioned in the task description (if any)

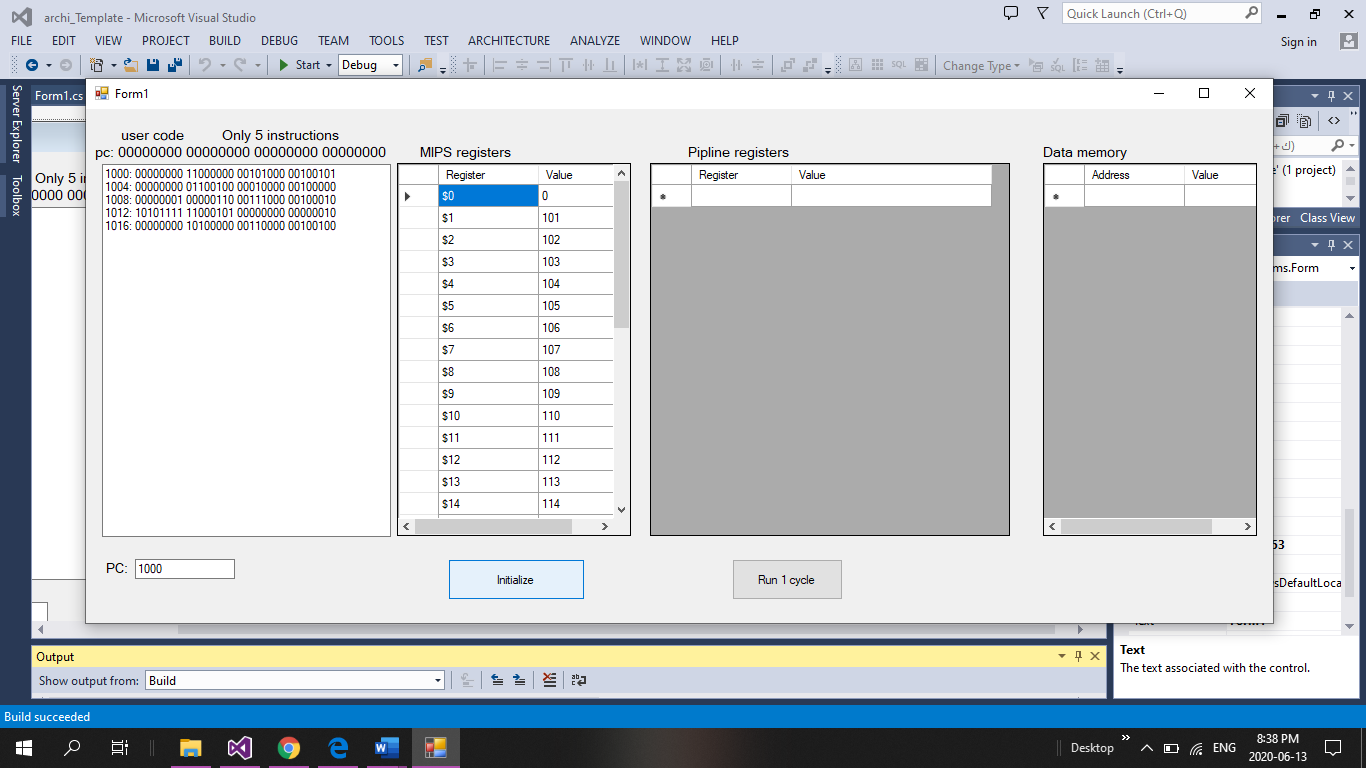
Code run only in 5 instructions to complete 9 cycles.

1. A user guide explaining how to use the emulator

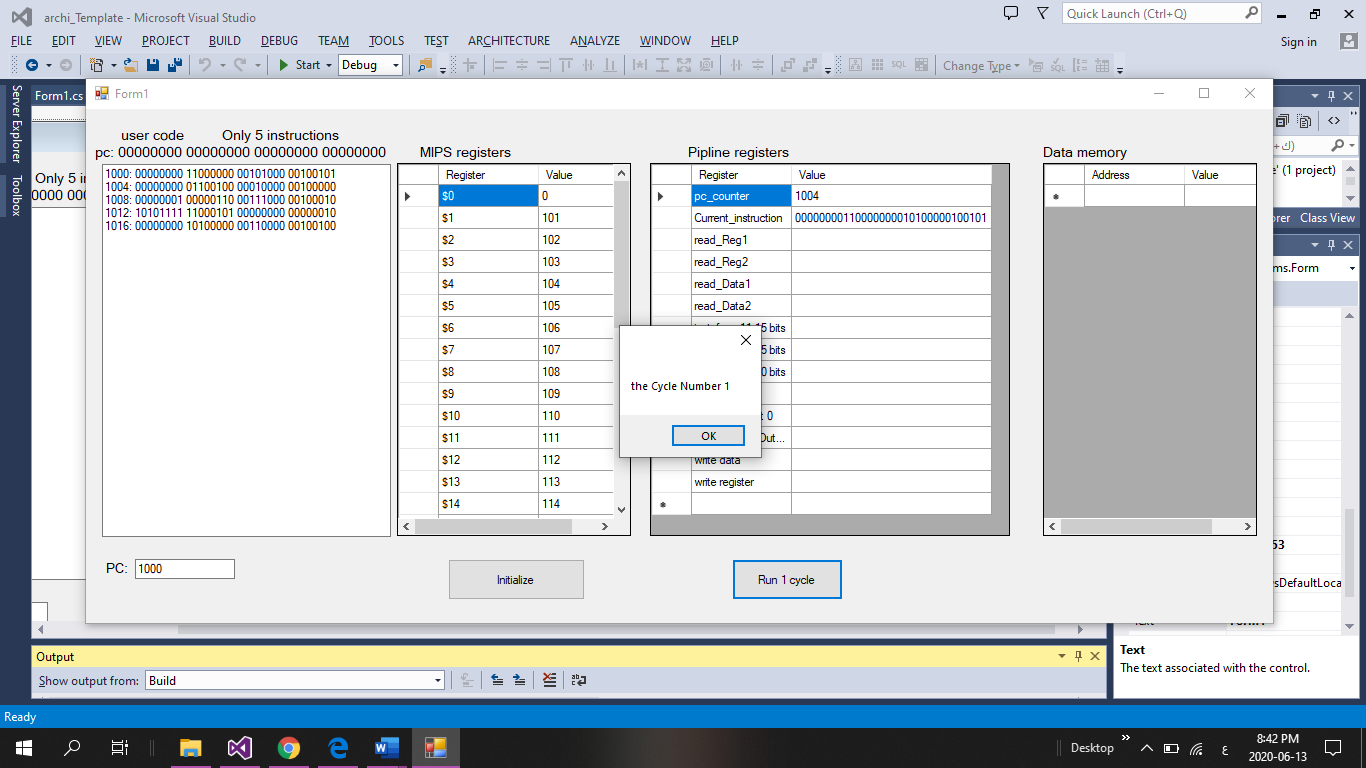
When start you can use the default instruction in textbox “user code” **OR** enter you own instruction but **only 5 instructions.** Run cycle button for next cycle.



**Use the default instructions**

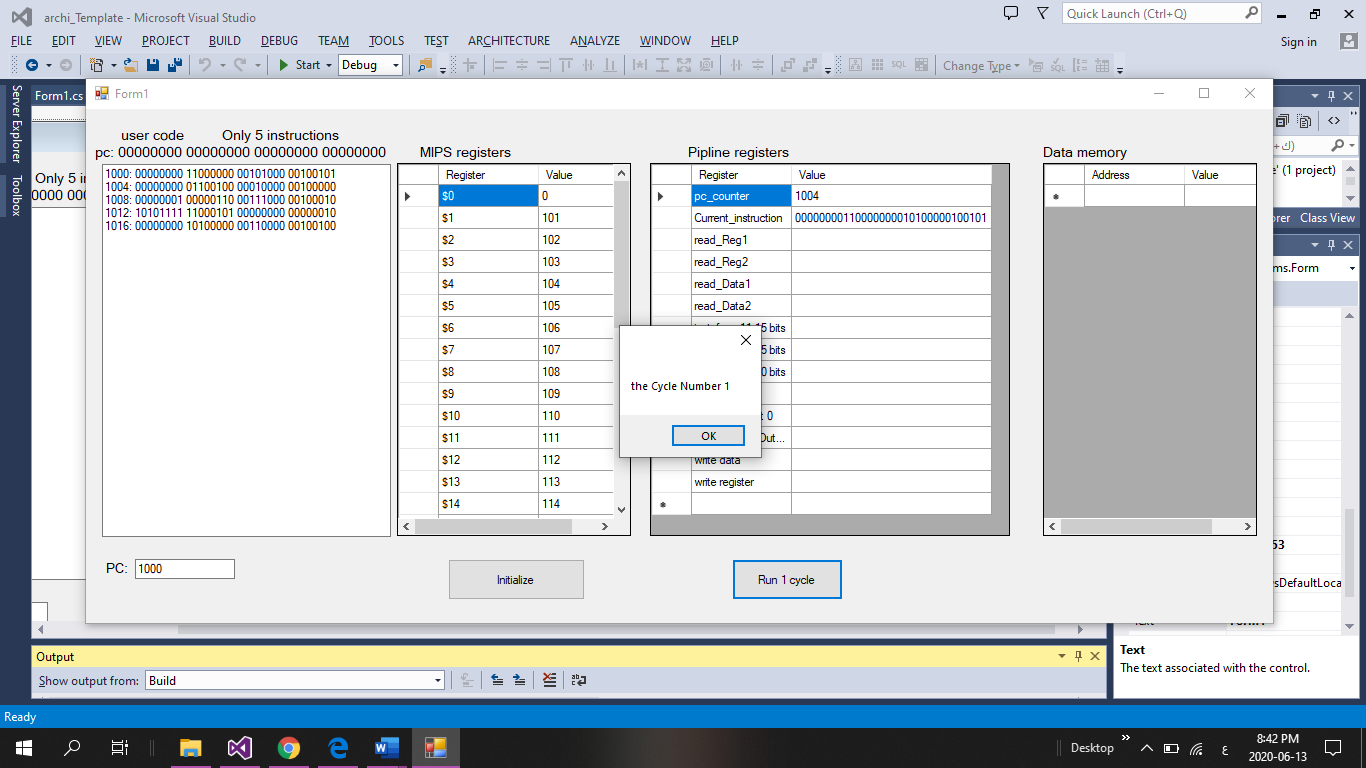


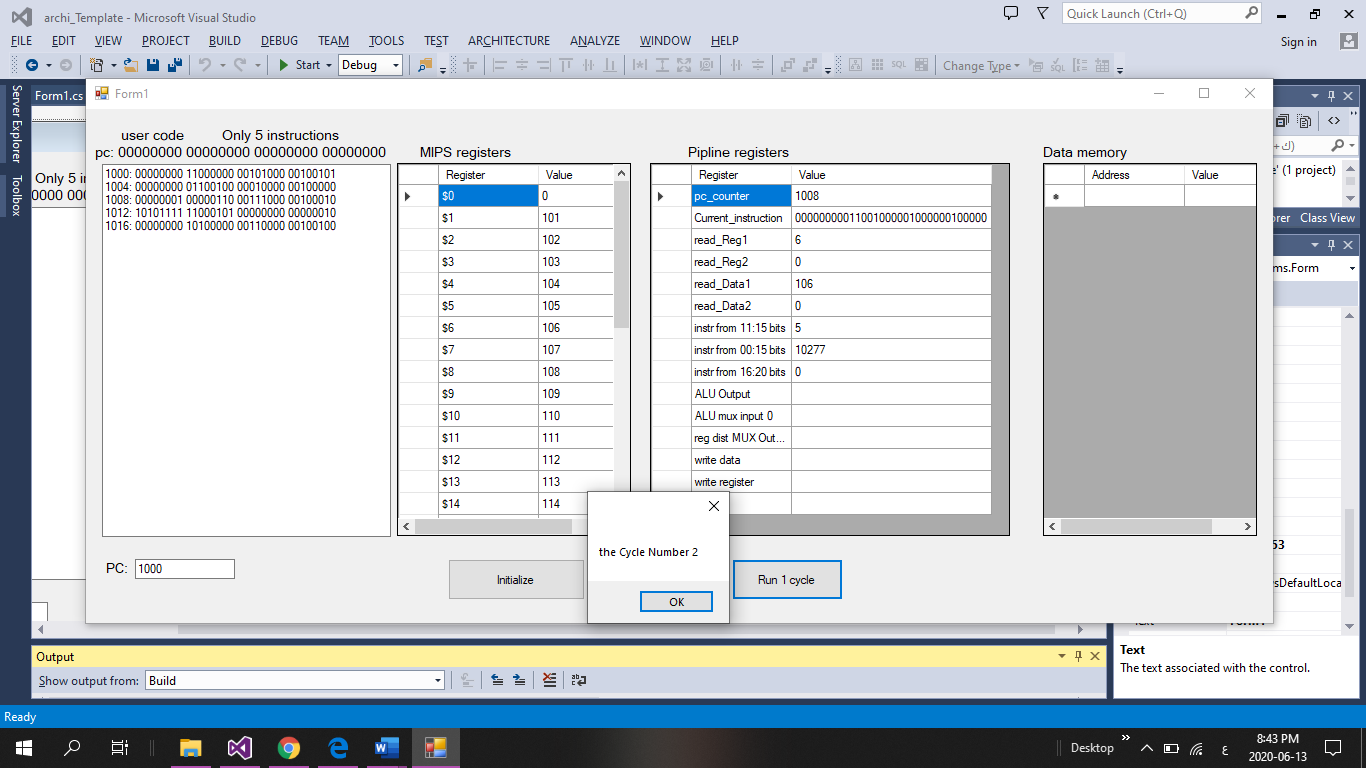
**Click initialize for initialize MIPS registers**

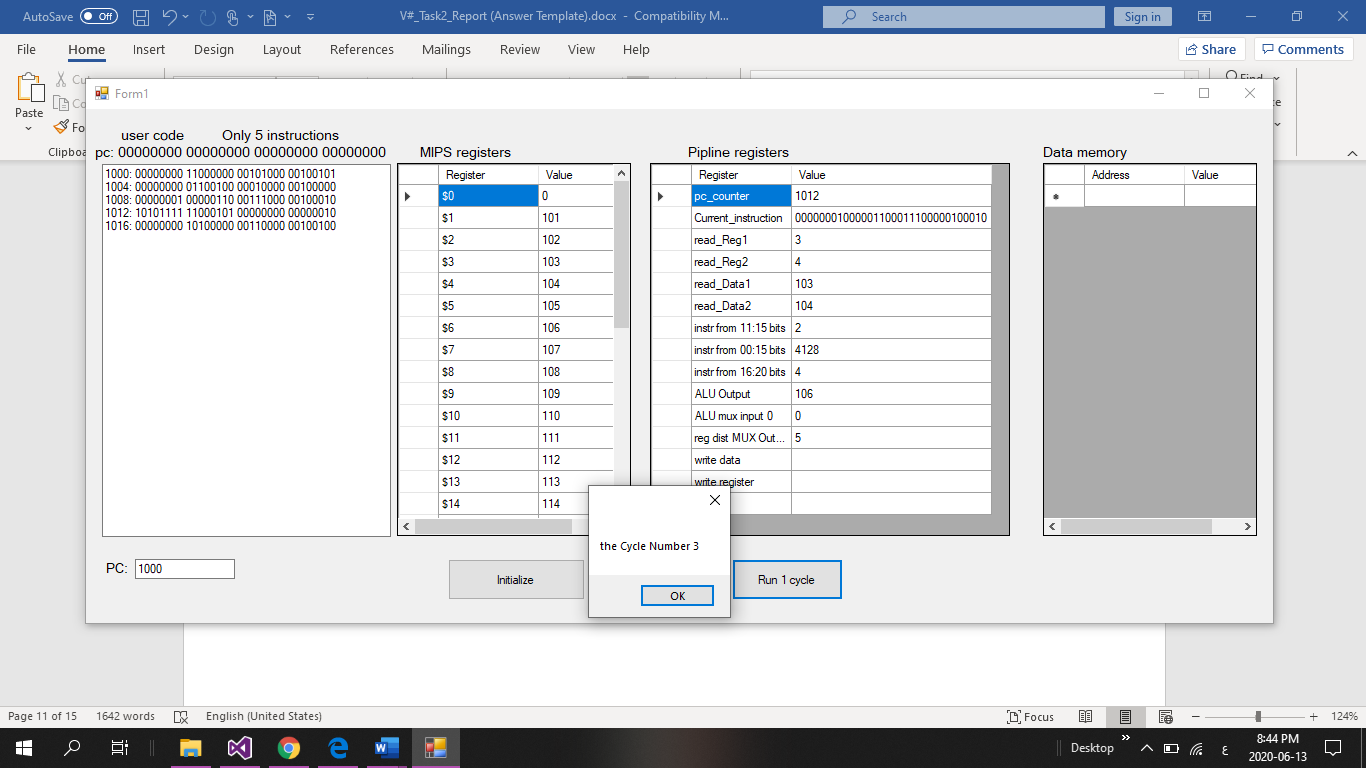


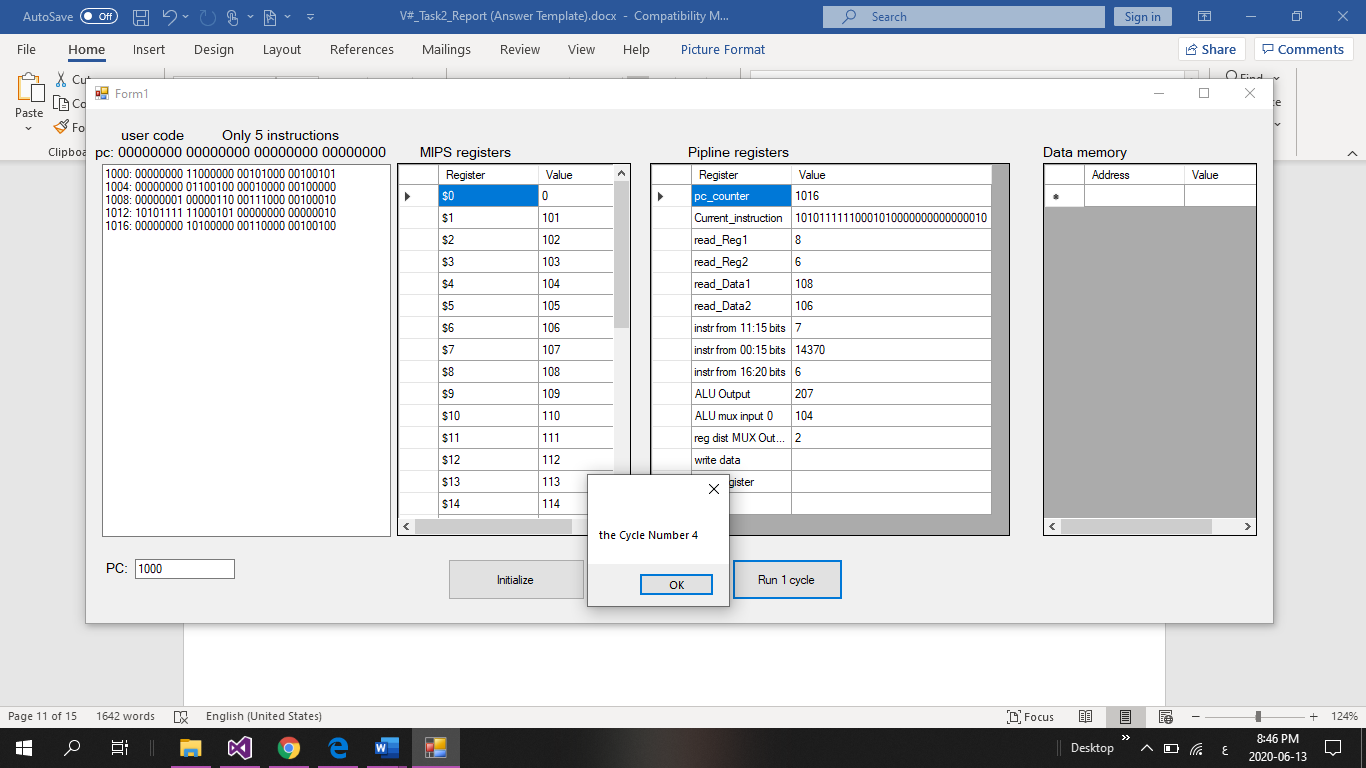
**Click run 1 cycle for execute cycle by cycle**

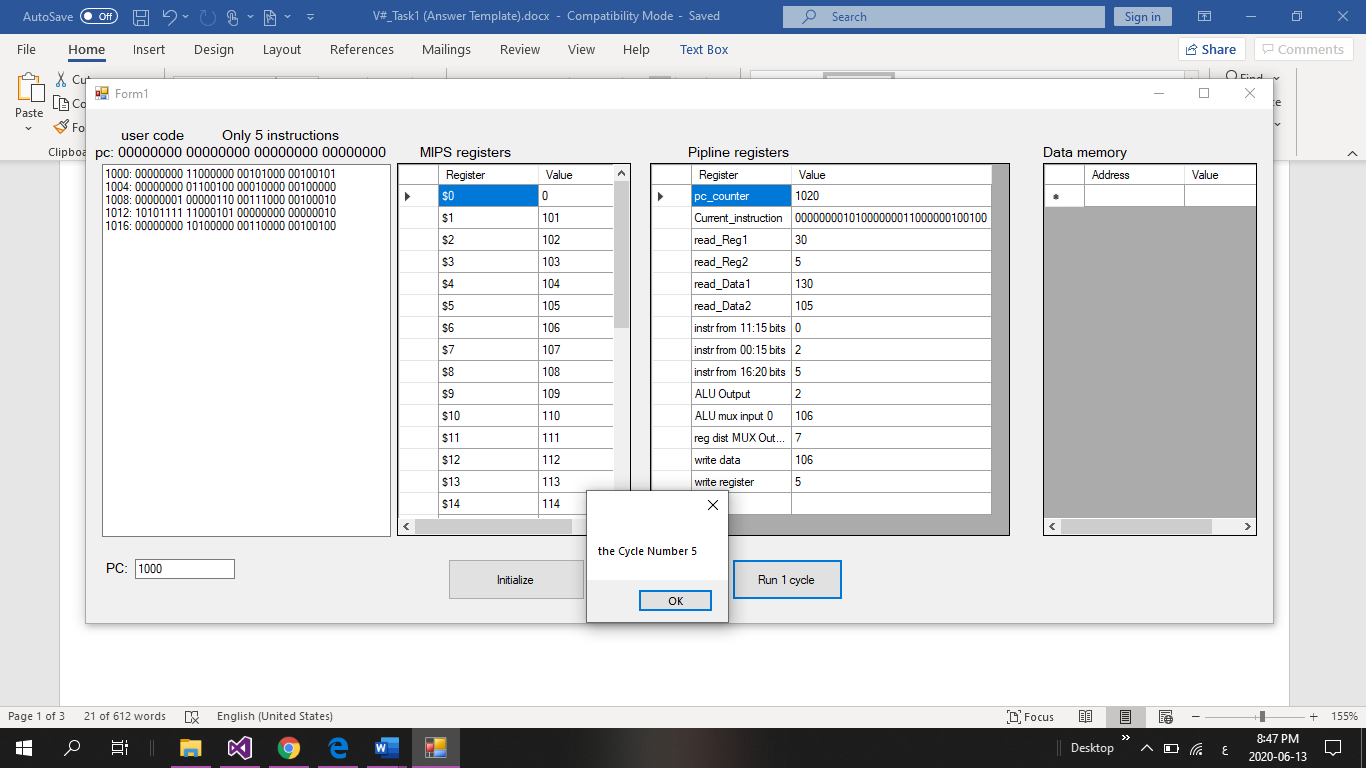
1. Screenshots of emulating the MIPS code given in Task 1, one screenshot per clock cycle for 9 clock cycles

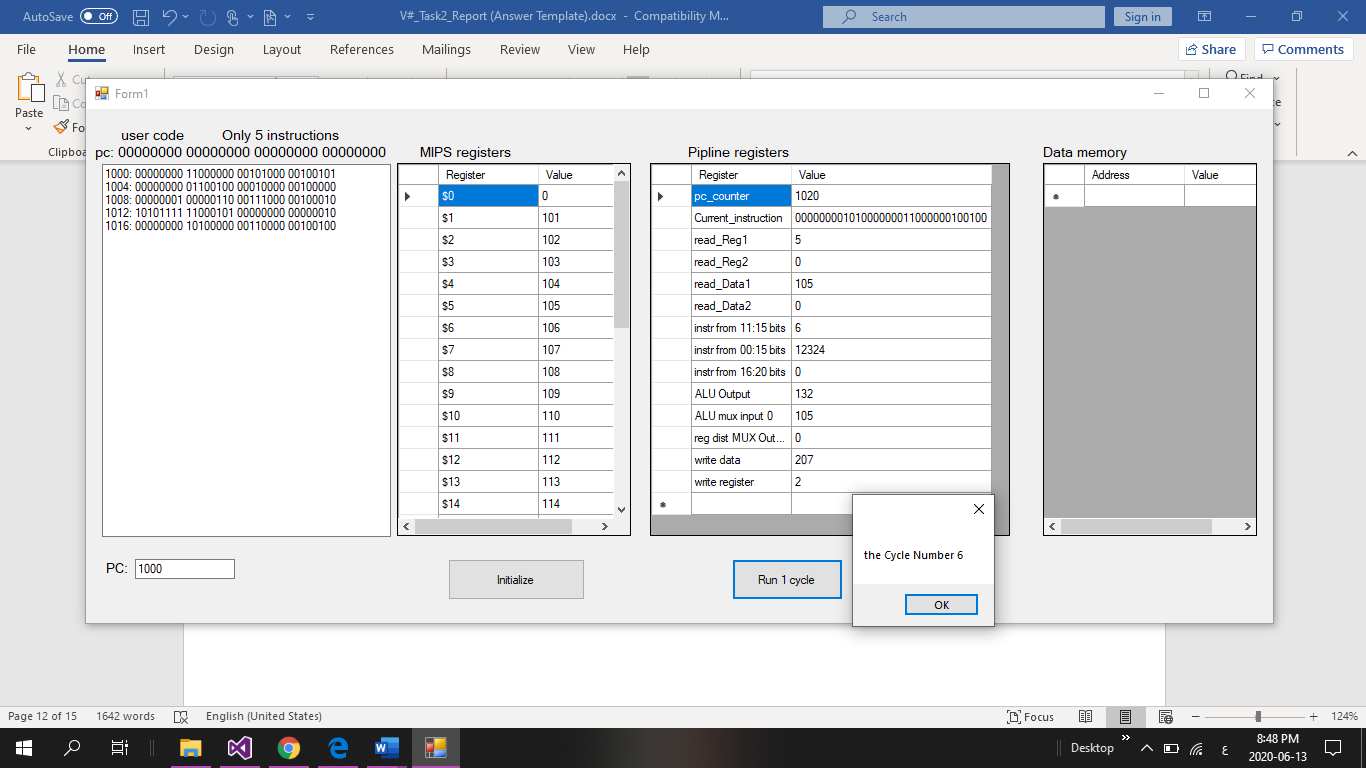


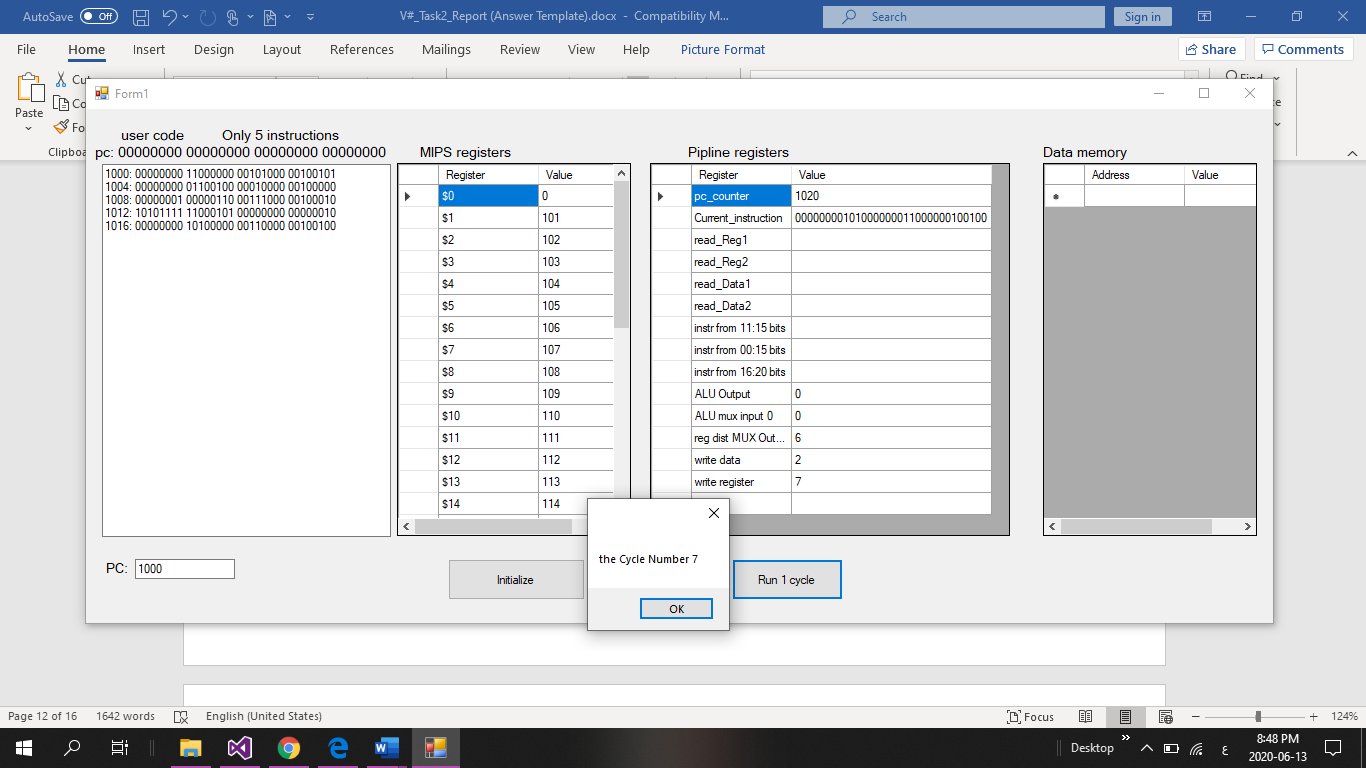


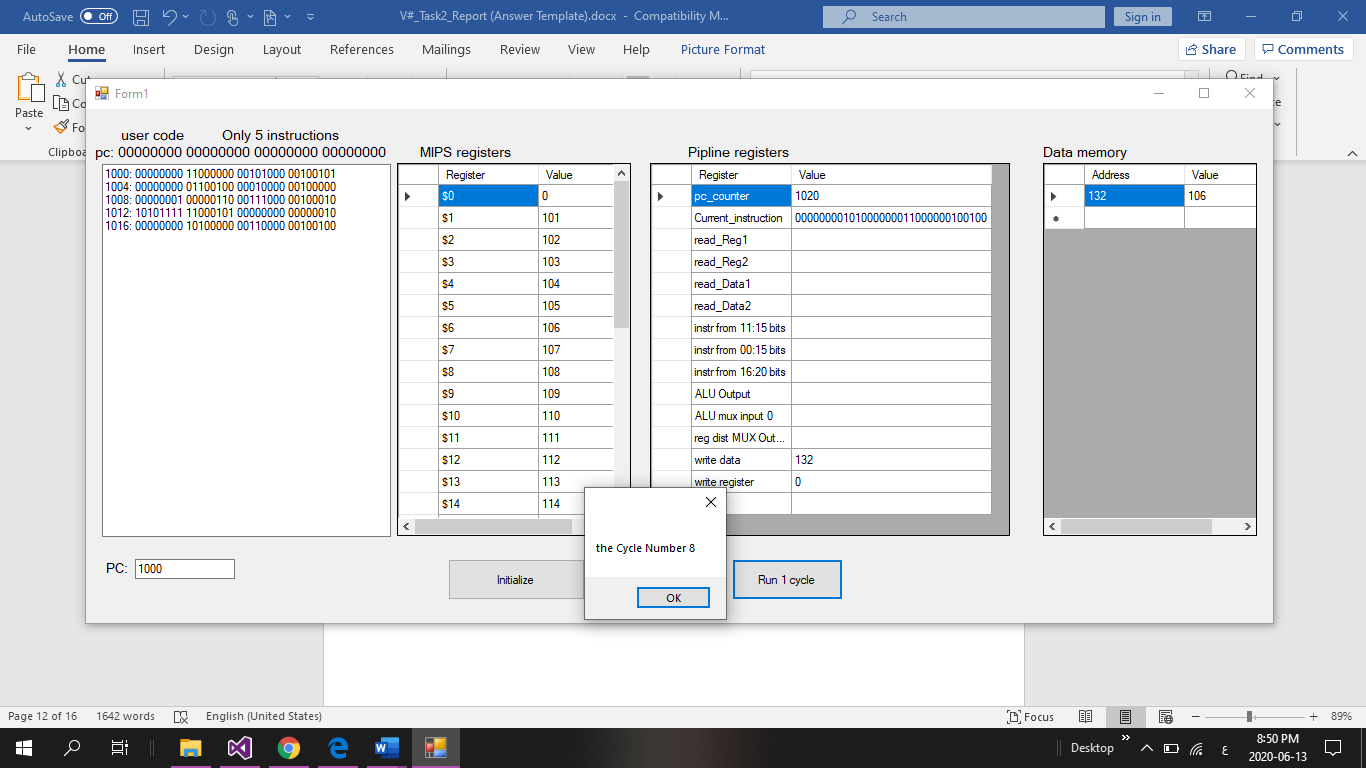




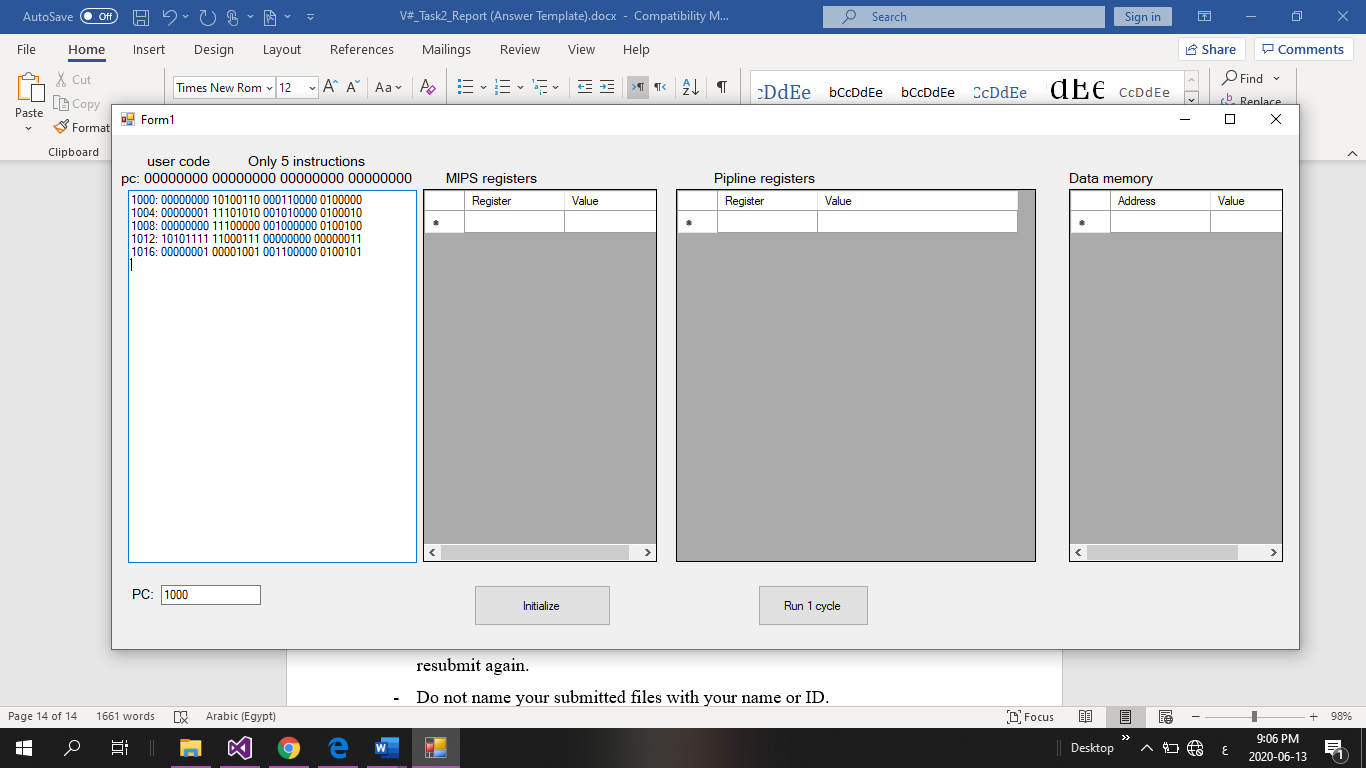


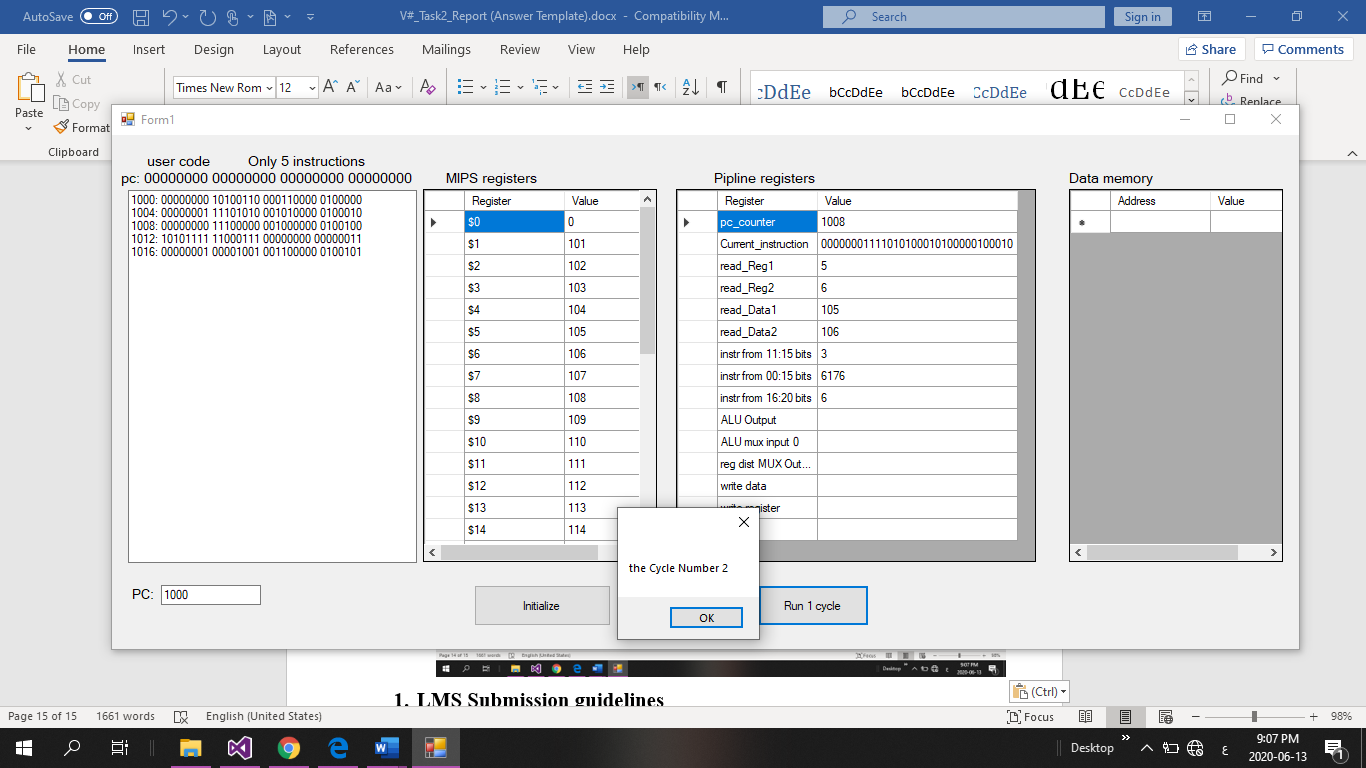
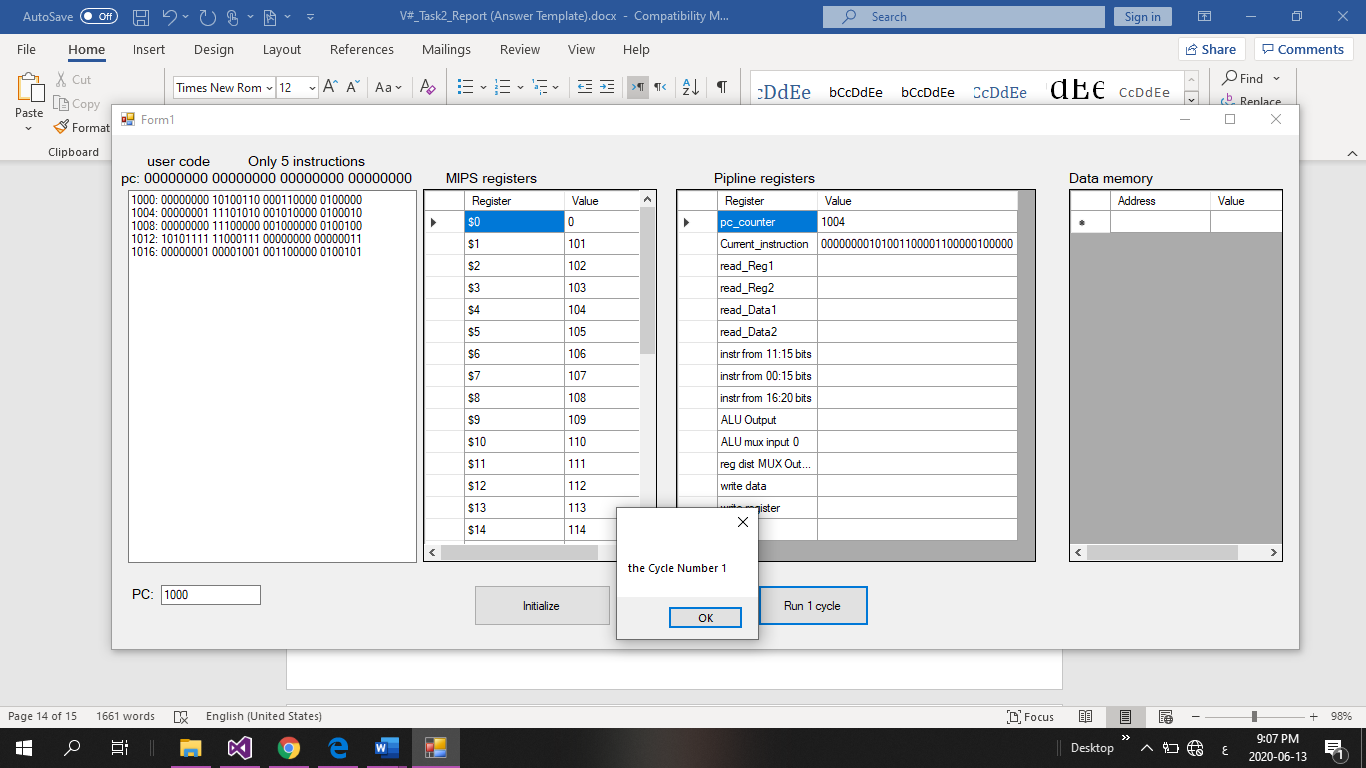


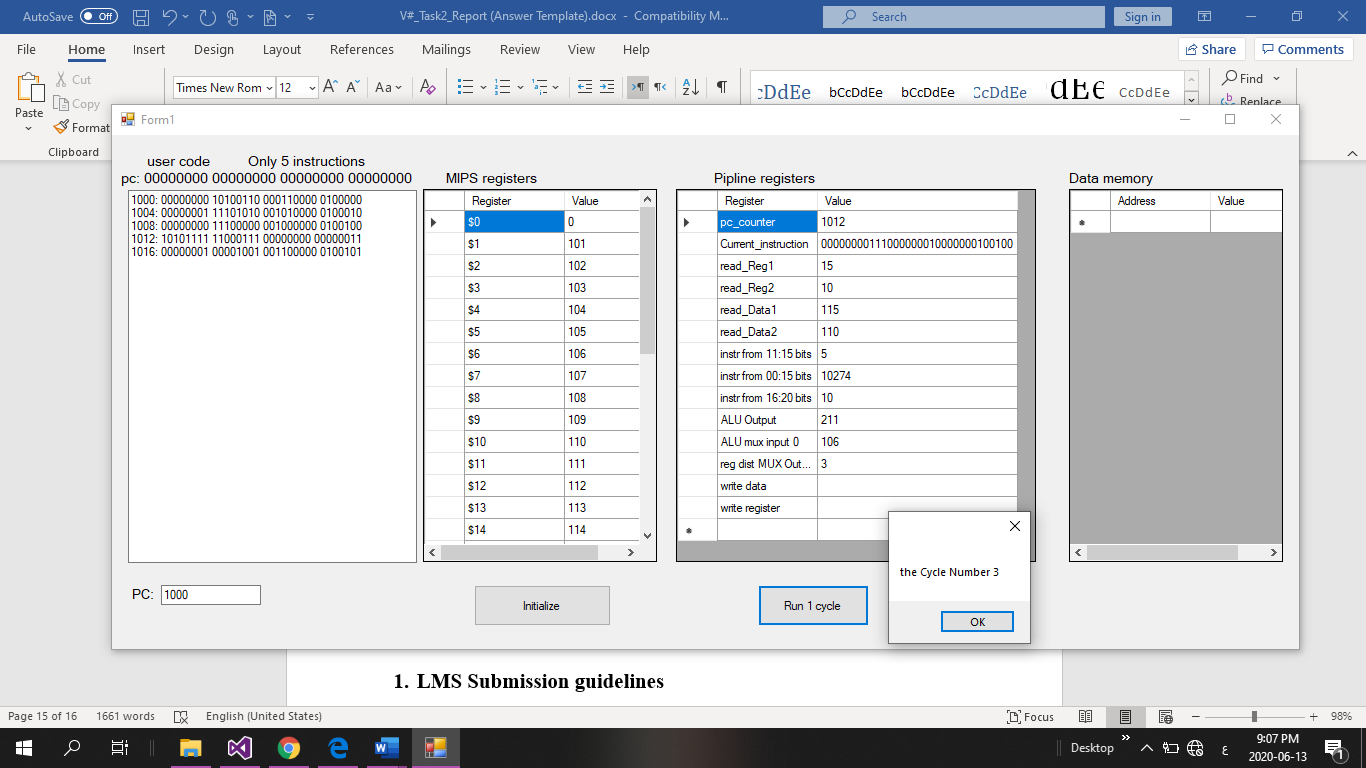


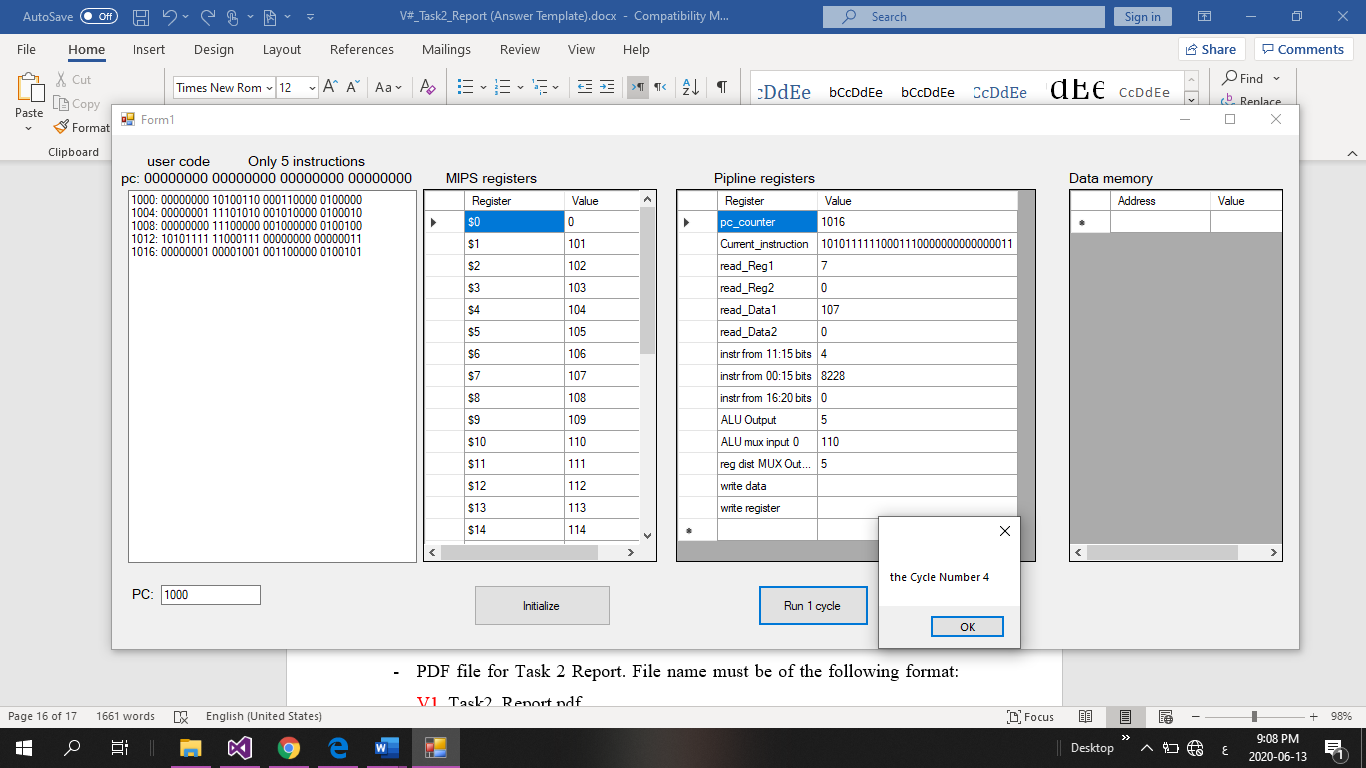
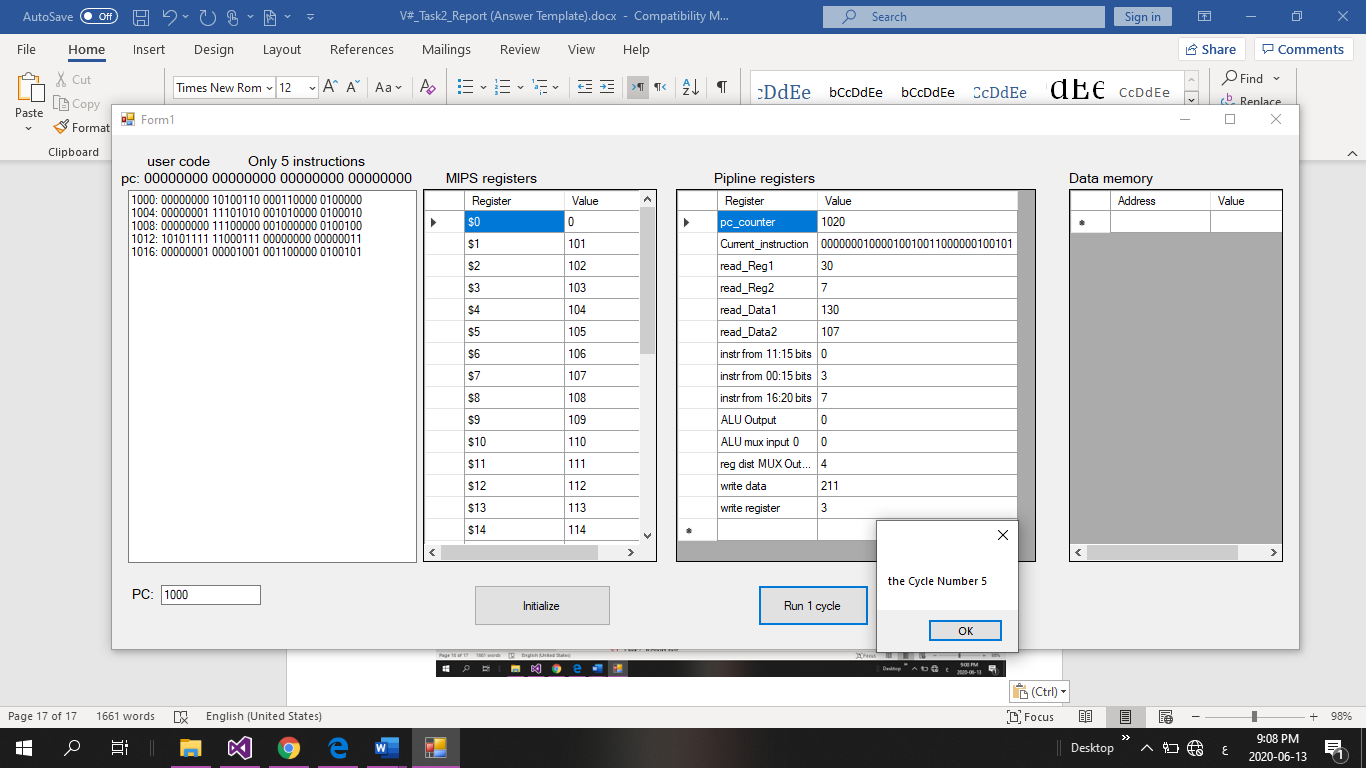


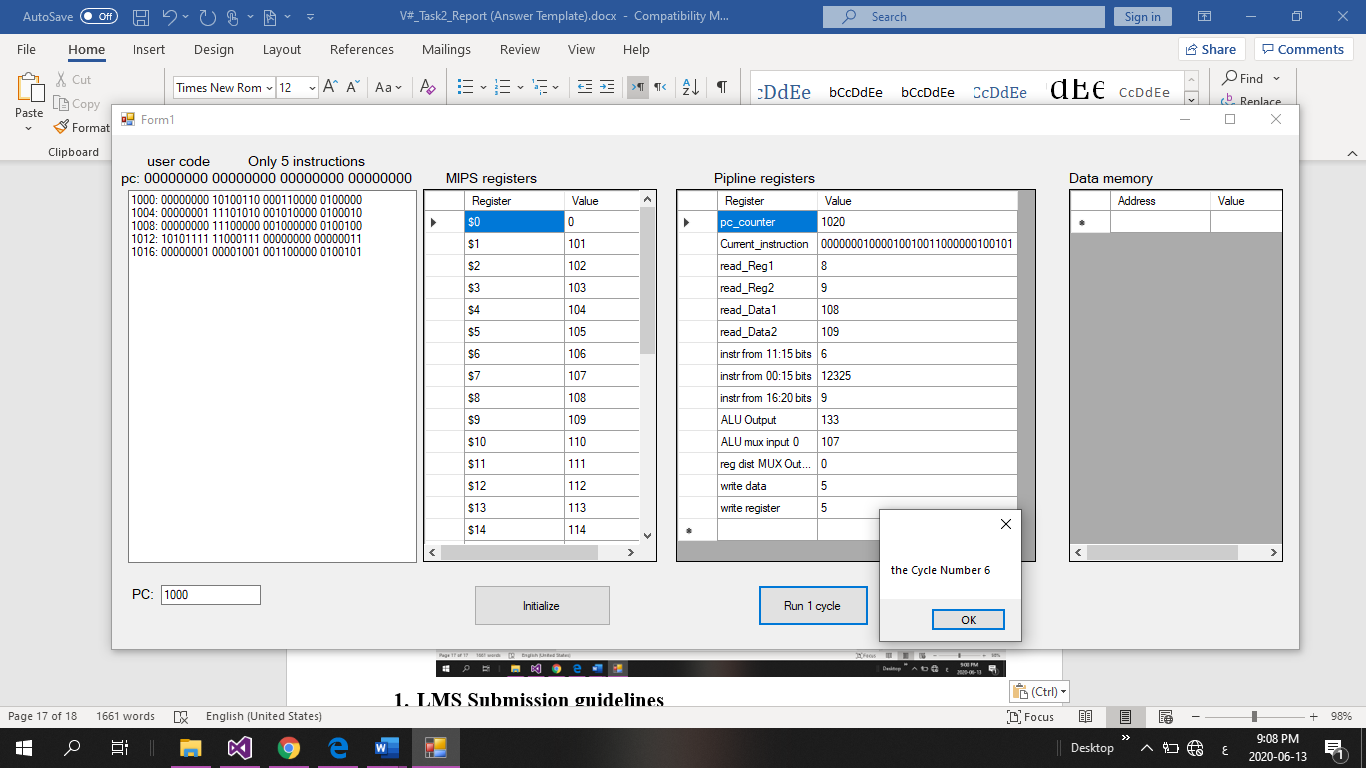
1. Screenshots of emulating another MIPS code from your choice, one screenshot per clock cycle for 9 clock cycles

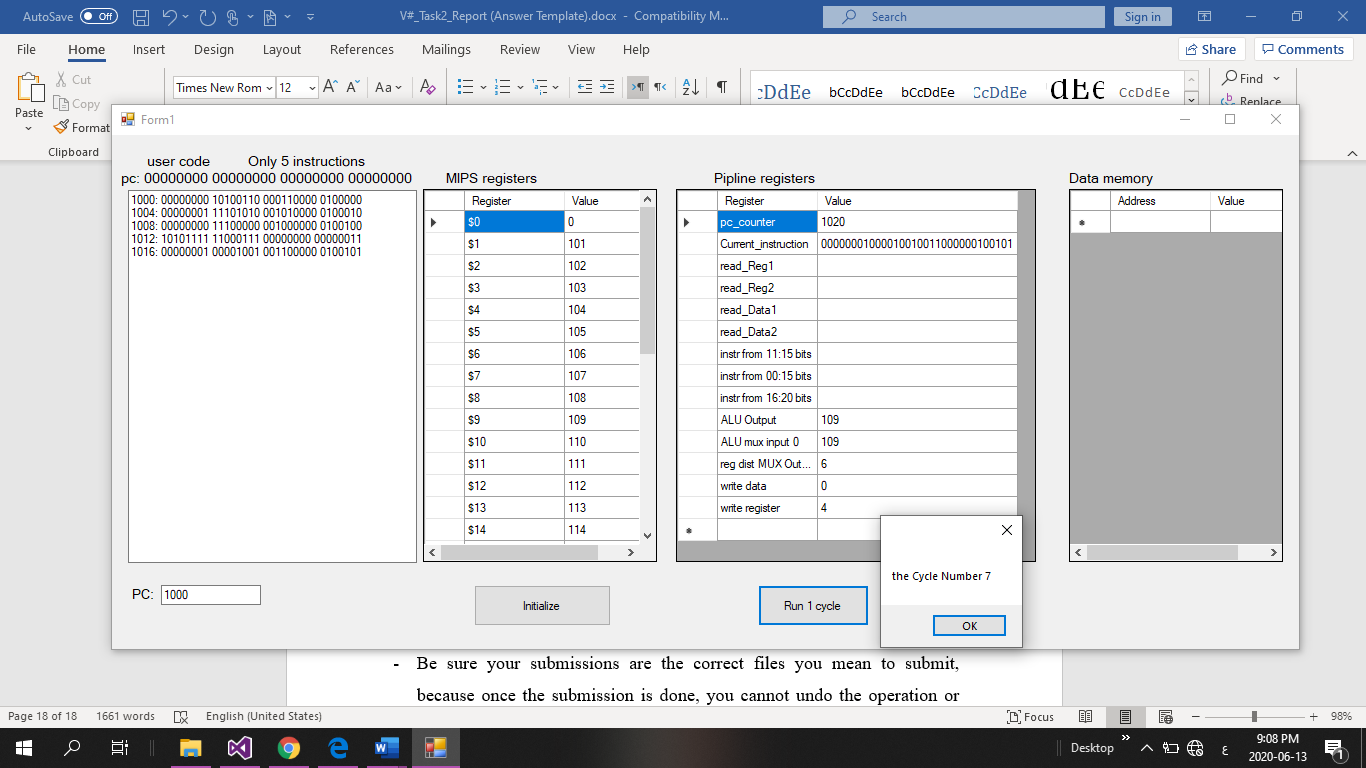


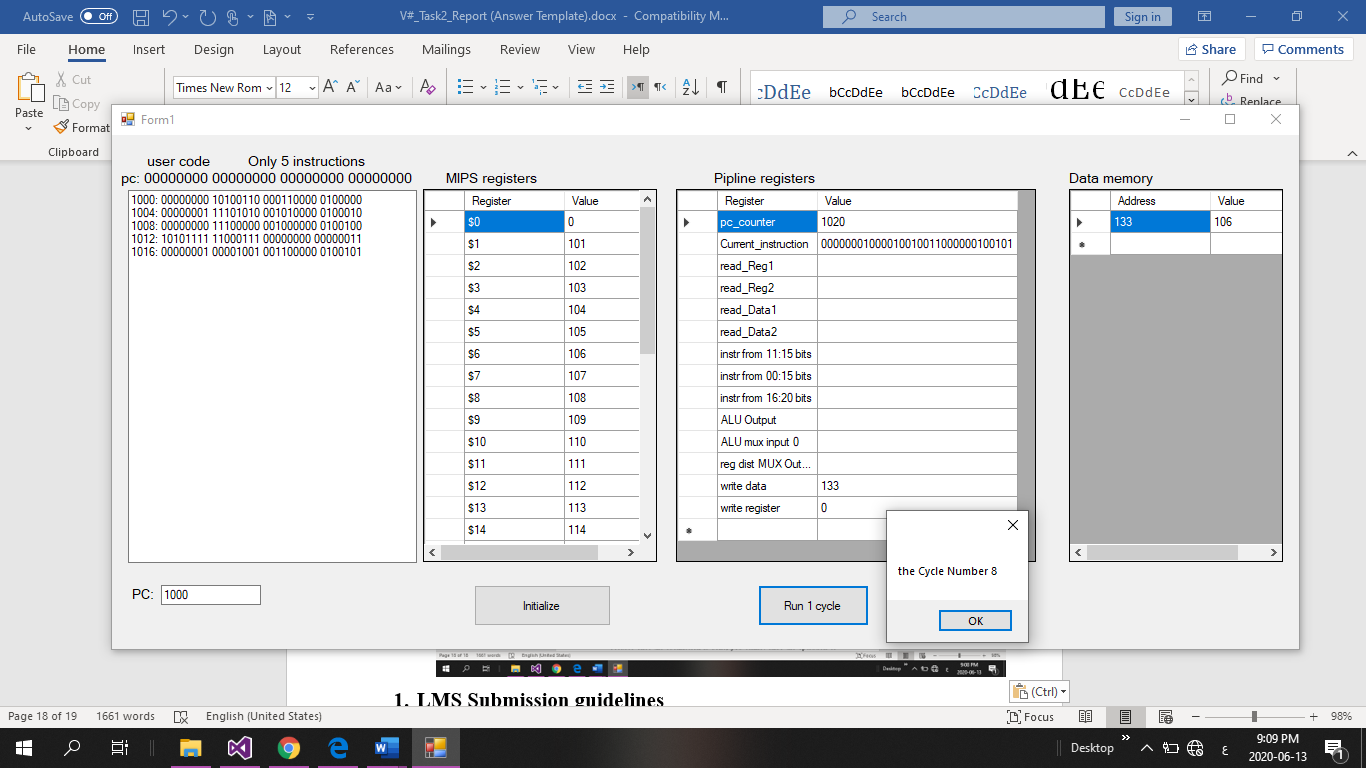


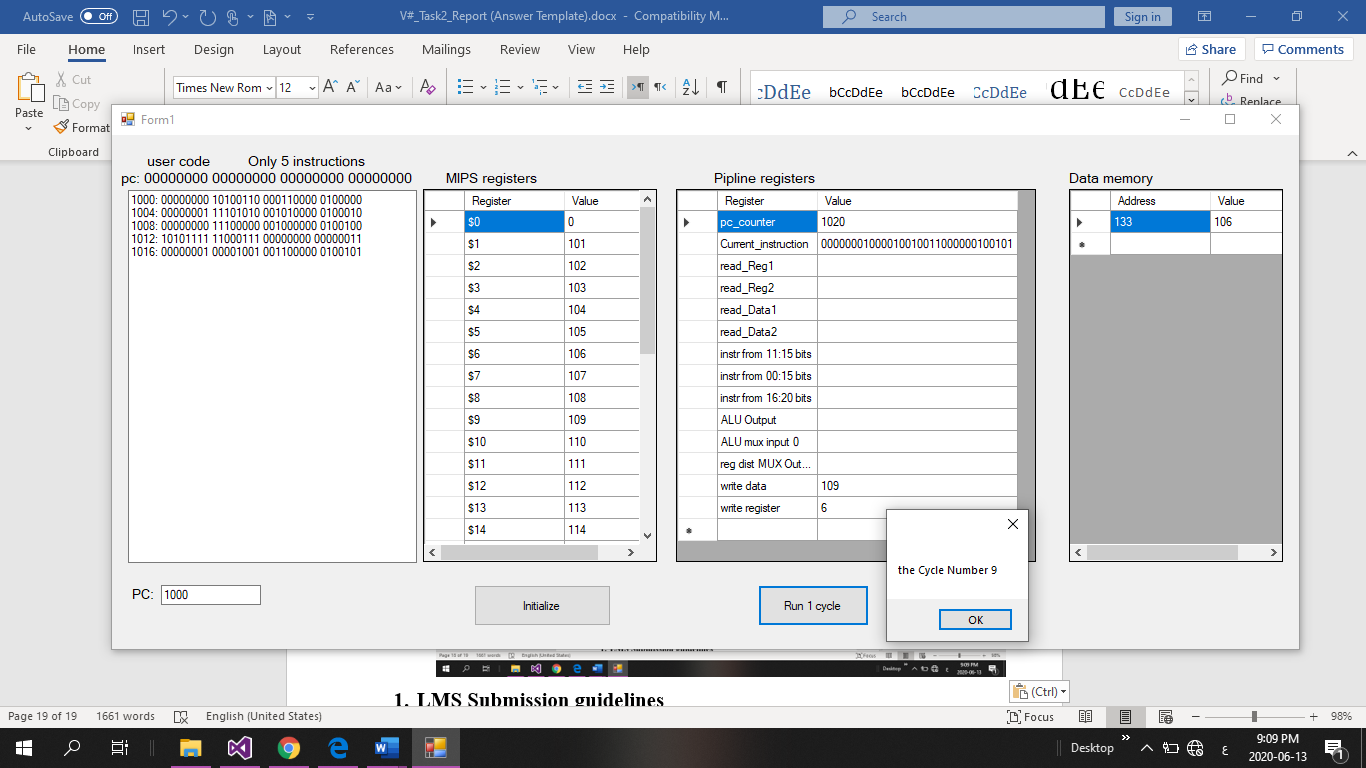












# **LMS Submission guidelines**

* Take care to submit your research not at the last minute, so as to meet the submission successfully without any tension.
* Be sure your submissions are the correct files you mean to submit, because once the submission is done, you cannot undo the operation or resubmit again.
* Do not name your submitted files with your name or ID.
* PDF file for Task 2 Report. File name must be of the following format: V1\_Task2\_Report.pdf

(V1 is for version1, use your exam version number after the letter V.)