```
#include "pipeline.h"
#include "instruction_memory.h"
#include "editor.h"
#include "registers.h"
#include "data_memory.h"
#include "5stages.h"
#include "RAW detection.h"
void PrintSpace(int num);
int count_pause_type_data_harzad();

int main() {
	cout << "Details of the ISA:" << endl;
	cout << "===================" << endl;
	cout << "It is a 32-bit ISA and can handle any 32-bit Arithmetic
operations" << endl;
	cout << "There are 32 GPRs <r0-r31>" << endl;
	cout << endl << endl;
	cout << "Select the opcode
'add','mov','sub','beq','mul','div','jmp','addi','subi','divi','muli' or
'end' to end code." << endl;
	cout << " And the instruction format is totally like MIPS" << endl;
	cout << "For example, 'add r1 r2 r3', or 'lw r3 5(r4)' or 'jmp 10' or
'addi r1 r2 3'" << endl << endl << endl;

	editor();

	DMem.InitializeDataMemory();
		Registers.InitializeRegisters();

	IF_ID.Initialization();
	ID_EX.Initialization();
	EX_MEM.Initialization();
	MEM_WB.Initialization();
	for (int i = 0; i < INSTRUCTION_MEMORY_SIZE; i++) {
		cyclesNum[i] = 5;
	}

	stages_5.IF();
	stages_5.ID();
	stages_5.EX();
	stages_5.MEM();
	stages_5.WB();
	int index_current = index - 1;


	while (IMem.im[Registers.PC] != 0) {

		stages_5.IF();
		stages_5.ID();


		index_current = index - 1;


		if (RAW_detection.forward_type_data_harzad_detection() ==
SOLVED_BY_FORWOARD) {
			harzardFlag[index_current][0] =
RAW_detection.forward_type_data_harzad_detection();
			harzardFlag[index_current][1] = HarzadRegNum;
		}
		else {
```

```
                  harzardFlag[index_current][0] =
RAW_detection.pause_type_data_harzad_detection();
                  harzardFlag[index_current][1] = HarzadRegNum;
            }

            if (RAW_detection.pause_type_data_harzad_detection() ==
SOLVED_BY_PAUSE) {
                  cyclesNum[index_current] = 6;
            }

            stages_5.EX();
            stages_5.MEM();
            stages_5.WB();

      }

      int total_cycles = 0;
      for (int i = 0; i < num_of_instructions_used; i++) {
            total_cycles += cyclesNum[i];
      }

      double  f = 500 * 1000000;

      double  clock_cycle = (1 / f) * 1000000000;
      double cpi = total_cycles / num_of_instructions_used;
      //CPU time
      double cpu_time = total_cycles * clock_cycle;
      double mips = f / cpi / 1000000;

      cout << "PC           Decoded£°           Encoded
instructions(32bit):      Clock cycles" << endl << endl;
      for (int i = 0; i<num_of_instructions_used; i++)
      {
            cout << "PC[" << instructions_used[i] << "]->    ";
            cout << instructions[instructions_used[i]] << ":     ";
            if (instructions[instructions_used[i]][0] == 'j') {
                  PrintSpace(7);
            }
            PrintAsBinary(IMem.im[instructions_used[i]]);
            cout << "    " << cyclesNum[i] << endl;
      }

      cout << endl << endl;

      cout << "After the program execution contents of the registers
are..............................." << endl << endl;
      Registers.PrintRegisters();
      cout << endl;
      cout << endl << endl;

      cout << "CPI of the program....................." << endl << endl;
      cout << "CPI =  " << cpi << endl;

      cout << endl << endl;

      cout << "MIPS of the program....................." << endl << endl;
      cout << "MIPS =  " << mips << endl;

      cout << endl << endl;

      cout << "CPU time of the program..............." << endl << endl;
```

```
	cout << "CPU time =  " << cpu_time << "  ns" << endl;

	cout << endl << endl;

	cout << "Pipelined Execution of the Program" << endl;
	cout << "==================================" << endl;
	cout << "It is assumed that the CPU has 5-pipeline stages: IF
(Instruction Fetch)" << endl;
	cout << "ID (Instruction Decoding), EX (Execution), MEM (Memory
Access) and WB (Write Back)." << endl;
	cout << "And each stage completes within one clock cycle." << endl <<
endl;
	cout << "The first kind of RAW hazard  (which means the output of ALU
is to be used as the next instruction's ALU's input)" << endl;
	cout << "will solved with FORWARDING (from the o/p  register of ALU "
<< endl;
	cout << "to the i/p of next instruction's ALU stage) without causing
any stall" << endl;
	cout << "The second kind of RAW  hazard (which means the output of
the Data Memory" << endl;
	cout << "is to be used as the next instruction's ALU's input) will
solved with a stall  " << endl << endl << endl;

	int num_display = num_of_instructions_used + 4 +
count_pause_type_data_harzad();
	cout << "                                ";
	for (int i = 1; i <= num_display; i++) {
		cout << i << "      ";
	}

	cout << endl;

	int stall_type_num = 0;
	for (int i = 0; i < num_of_instructions_used; i++) {

		cout << i << ".  ";
		cout << instructions[instructions_used[i]] << ":";

		if ((i >= 2) && (harzardFlag[i - 2][0] == 2))
		{
			PrintSpace(8);
		}
		else {
			PrintSpace(11);
		}


		if (instructions[instructions_used[i]][0] == 'j') {
			PrintSpace(6);
		}


		PrintSpace(i * 8);


		if ((stall_type_num == 0) || ((i >= 2) && (harzardFlag[i -
2][0] == 2)) || ((i >= 1) && (harzardFlag[i - 1][0] == 2)))
		{
		}
		else {
			PrintSpace(6 * stall_type_num);
```

```
            }
            if ((i >= 2) && (harzardFlag[i - 2][0] == 2))
            {
                  cout << " stall   ";
            }

            cout << "IF  | ";

            if ((i >= 1) && (harzardFlag[i - 1][0] == 2))
            {
                  cout << " stall ";
                  cout << "|";
            }

            cout << "  ID  | ";


            if (harzardFlag[i][0] == 2)
            {

                  cout << " stall ";
                  cout << "|";
                  stall_type_num += 1;
            }

            cout << " EX  |  MEM  |  WB       ";

                        cout << endl;
      }

      cout << endl << endl;

      cout << "RAW hazard details:" << endl;
      cout << "====================" << endl;

            bool flag = false;
      for (int i = 0; i < num_of_instructions_used; i++) {
                  if (harzardFlag[i][0] == 1) {
                        flag = true;
                        cout << "r" << harzardFlag[i][1] << " in
instruction " << i - 1 << " and " << i << " caused RAW hazard and is solved
by forwarding" << endl;
                  }
                  else if (harzardFlag[i][0] == 2) {
                        flag = true;
                        cout << "r" << harzardFlag[i][1] << " in
instruction " << i - 1 << " and " << i << " caused RAW hazard and is solved
by stall" << endl;
                  }
                  else {
                        continue;
                  }
      }
      if (flag == false) {
                  cout << "There's no RAW hazard detected." << endl;
      }

      cout << endl << "Pipelined execution took  " << total_cycles << "
clock cycles for the program execution." << endl << endl;

      system("PAUSE");
```

```
}

void PrintSpace(int num) {
	for (int i = 0; i < num; i++) {
		cout << " ";
	}
}

int count_pause_type_data_harzad() {
	int count = 0;
	for (int i = 0; i<num_of_instructions_used; i++)
	{
		if (harzardFlag[i][0] == 2) {
			count += 1;
		}
	}
	return count;
}
```