Task: Implement a MIPS Emulator

Task 1 is tedious to be solved manually, isn't it? Therefore, you are required to develop an emulator for MIPS pipelined CPU which can show the values of internal CPU components cycle by cycle for any given machine code. Your emulator will only support the following MIPS instructions: **add, sub, lw, and, or.**

The user should be able to input the machine code of a MIPS program (using the supported instructions only) and specify its starting address by setting the PC register value.CPU registers are initialized as specified in Task 1 and all data memory words are assumed to be initialized by value 99. The emulator should display any memory word used by the MIPS program.

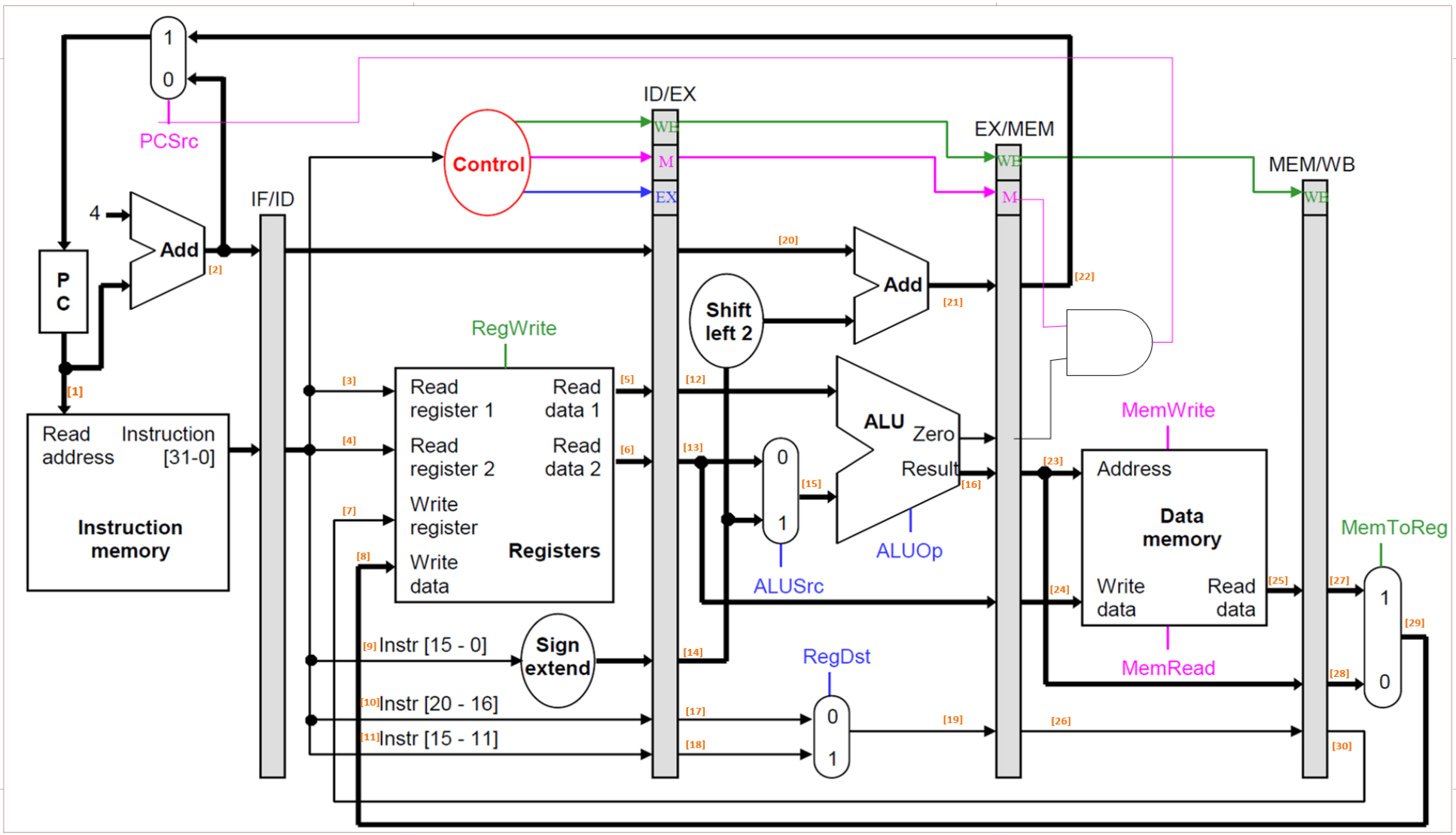
Note: You are not required to handle any pipeline hazards. (The input MIPS code should not contain any hazards).

Implementation Hints:

- Define an array for MIPS registers (0-31) and define variables for pipeline registers. Make them class data members to be accessible from any class function.
- Define 2 hash tables for instruction and data memories where the memory address is treated as the hash table key (unsigned int) and memory word is treated as the hash table value (32 characters string). Fill the instruction memory by the binary codeof the user program at the specified addresses.
- Organize your code so that MIPS components (e.g., register file, MUXs, Adders, etc.) are represented as functions where they read from specific class variables and update other class variables according to the MIPS data path (Figure 1). Remember that these class variables are emulating MIPS registers.
- MIPS control unit and ALU can be implemented by if/else statements instead of logic gates. AND and OR operations can be done in C++ and C# using & and | operators, respectively.
- At each clock cycle, you only need to call all these functions to update the CPU state and move the execution forward.
- Fill the instruction memory by 4 or more NOP instructions after the end of user program to avoid possible bugs when trying to read empty memory. The machine code of NOP is all zeros.

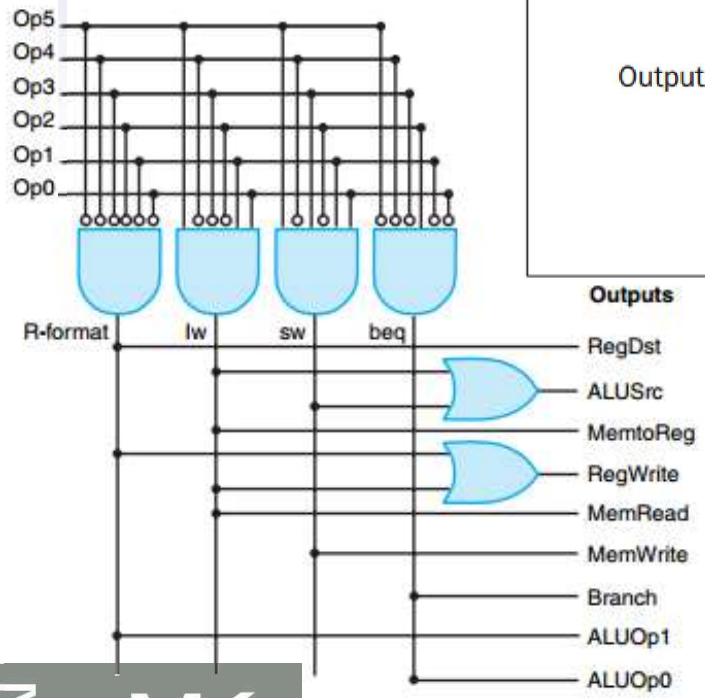
Requirements:

1. The user should enter a MIPS machine code with its address in a textbox. Code should consist of the supported instructions only.
2. When the user presses "Initialize", the program resets to its initial state as specified in Task 1: Give initial values to PC, MIPS registers, data memory, reset pipeline registers, update the GUI lists accordingly and fill the emulated instruction memory (hash table) with the given code in the textbox.
3. When the user presses "Run 1 Cycle", the program runs the emulated MIPS components to move pipeline stages forward and updates the GUI accordingly.


1
0
PCSrc
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
4
Add
[2]
P
C
IF/ID
[1]
Read address
Instruction [31-0]
Instruction memory
RegWrite
[3]
[4]
[5]
[6]
[7]
[8]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
[9] Instr [15 - 0]
[10] Instr [20 - 16]
[11] Instr [15 - 11]
Sign extend
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]
[20]
[21]
[22]
Shift left 2
Add
ALU
Zero
Result
ALUOp
ALUSrc
RegDst
MemWrite
[23]
[24]
[25]
Address
Data memory
Write data
Read data
MemRead
MemToReg
[26]
[27]
[28]
[29]
[30]

Control Unit Example

Control	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



ALU Control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111