MIPS Interactive Simulator

https://github.com/decltypeme/mips-simulator

Open source C++ pipelined MIPS simulator that implements a subset of the MIPS ISA. It features memory, registers, pipeline viewers, and easy to use GUI.

Hussam Ashraf El-Araby

Islam Faisal Ibrahim Abdelrasoul

Mohammed Hassan Ismail

13th December 2015

Project Overview

This simulator was done as part of our Computer Architecture coursework.

It is a simulator of a 5-stage pipelined MIPS processor.

It supports the following features:

- Parsing of input instructions to check for syntax errors.
- 5 stage pipelined MIPS CPU (data path, control unit, 32 registers, instruction memory, data memory)
- Support the following instructions
    - ADD,ADDI,XOR,LW,SW, BLE, J, SLT, JAL, JR
    - JMP
        - Jump procedure
            - relative jump to address in immediate
            - pushes return address to stack
    - RET
        - Return procedure
            - pops top address in stack
            - jumps to it
- Hazard detection and dealing with different hazards (explained more in the next section)
- Call stack of depth 4
- Exceptions are handled and errors are reported with location and reason.

The GUI adds the following features:

- View parsed assembly instructions, PC, registers, exceptions and other messages
- View up to 16 data memory locations
- View call stack
- View state of current pipeline and history for 9 previous cycles
- View hazards that happen in each cycle and what was done to avoid the hazards
- Buttons to load source and reset simulator.

Note 1: The application .exe and regex rule file are included in the submission folder.

Note 2: Code is pasted in Appendix A for reading.

To access the code easily and without copy-paste errors use the GitHub repo @ https://github.com/decltypeme/mips-simulator

Note 3: The test files and any attached images or files are included in the submission folder.

Parsing Method

To check the syntax of a given instructions file and parse it, we use regular expressions. First a high level description of the syntax is provided (not in regex yet) as below.

```
({r-inst})\s+{reg}\s*,\s*{reg}\s*,\s*{reg}\s*{EOL}#R-InstructionTAKECARE-In-New-Implementation-We-Disallow-Comments

({i-inst})\s+{reg}\s*,\s*{reg}\s*,\s*{imm}\s*{EOL}#I-Instruction

({j-inst})\s+{imm}\s*{EOL}#J-Instruction

({jr-inst})\s+{reg}\s*{EOL}#JR-Instruction

({ret-inst})\s*{EOL}#RET-Instruction

\s*{EOL}#Empty-Instruction
```

Listing 1: Syntax Outline Example

Then, we create a file that describes the major parts in our syntax like {r-inst} or {EOL}, then using a tool we developed, the file is transformed into a set of well-defined regex rules.

{r-inst}	[Aa][Dd][Dd]|[Xx][Oo][Rr]|[Ss][Ll][Tt]
{i-inst}	[Aa][Dd][Dd][Ii]|[Ll][Ww]|[Ss][Ww]|[Bb][Ll][Ee]
{j-inst}	[Jj][Aa][Ll]|[Jj]|[Jj][Mm][Pp]
{jr-inst}	[Jj][Rr]
{ret-inst}	[Rr][Ee][Tt]
{reg}	\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])
{imm}	(0x[0-9A-Fa-f]+|(?:0d){0,1}[0-9\-]+)
{EOL}	(?:\#(?:.*)){0,1}

Listing 2: The replace rules

```
([Aa][Dd][Dd]|[Xx][Oo][Rr]|[Ss][Ll][Tt])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*,\s*\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*,\s*\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*(?:\#(?:.*)){0,1}

([Aa][Dd][Dd][Ii]|[Ll][Ww]|[Ss][Ww]|[Bb][Ll][Ee])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*,\s*\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*,\s*(0x[0-9A-Fa-f]+|(?:0d){0,1}[0-9\-]+)\s*(?:\#(?:.*)){0,1}

([Jj][Aa][Ll]|[Jj]|[Jj][Mm][Pp])\s+(0x[0-9A-Fa-f]+|(?:0d){0,1}[0-9\-]+)\s*(?:\#(?:.*)){0,1}

([Jj][Rr])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*(?:\#(?:.*)){0,1}

([Rr][Ee][Tt])\s*(?:\#(?:.*)){0,1}

\s*(?:\#(?:.*)){0,1}
```

Listing 3: The final regex rule describing the syntax of code files

Hazard Detection

The hazard detection process happens during each single cycle.

A function is invoked in our simulator that returns what hazards has been detection then the datapath component of the simulator invokes other functions that deal with those hazards.

Hazards include:

- Read after write
- Read after load
- Jump or branch instruction, interrupting the normal flow of instructions
- Wrong prediction of a branch, interrupting the normal flow of instructions

Hazard solutions:

- Forwarding:
    - Due to:
        - Read after write
        - Read after load
    - Between any of
        - {execute, memory, write back stages}
            - to any of {decode, execute, memory, stages}

- Flushing
    - Due to:
        - Jump or branch instruction
            - Flush the fetched instruction
        - Wrong prediction of a branch
            - Flush the fetched and decoded instruction

Note: This simulator does the branch prediction in the decode stage. A better implementation would be doing it in the fetch stage.

User Interface

The interface used in the program is described below. Moreover, the main program functions are described through the description of the buttons.

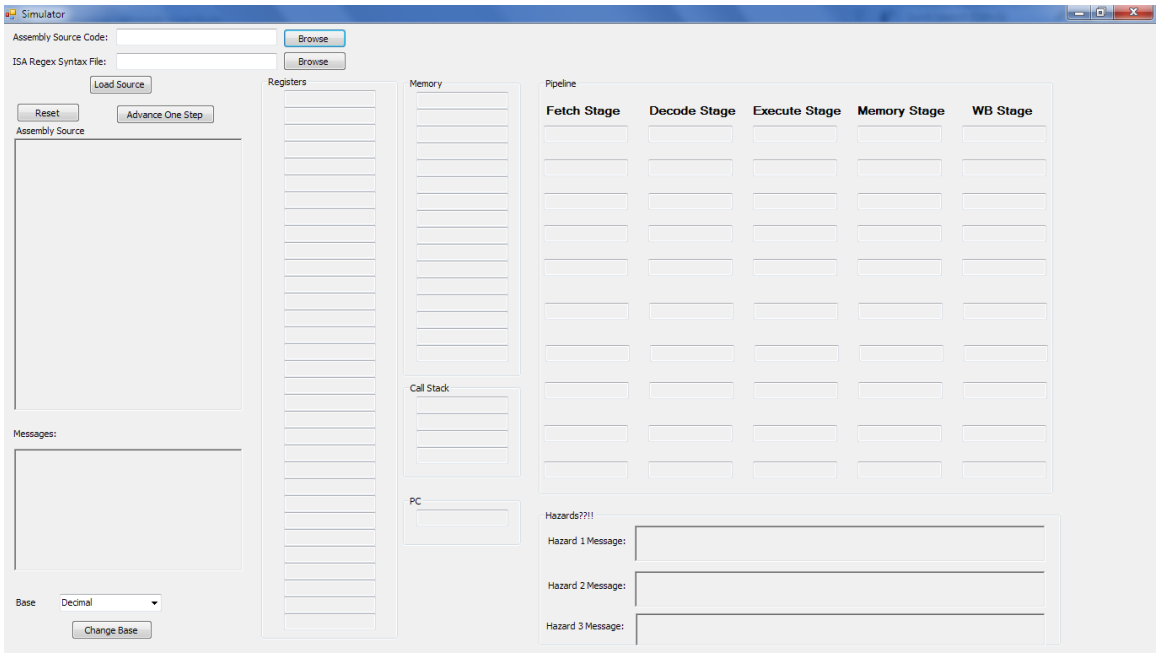


Figure 1 User Interface of the project

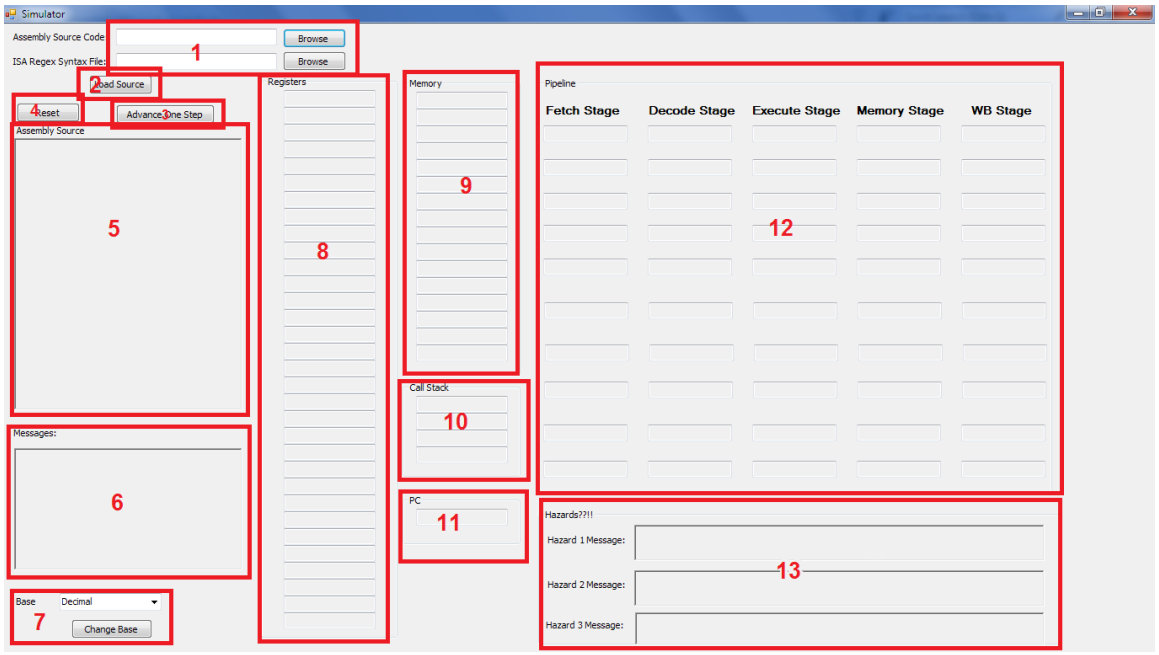


Figure 2 Annotated User Interface

1: *Browse files* textboxes to input the filename of the MIPS program to run and the filename of the regular expression rules to be used.

2: *Load source button*: Once pressed it attempts the following (producing errors if any process faces an error)

1. Do what the reset button does
2. In addition, fill the instruction memory with nops.
3. Open the program file and the regex file
4. Send both files to the regex parser
5. Parse all program instructions and check that all of them are syntactically correct
6. Dynamically construct the instructions and copy pointers to the instructions to the data structure that represent the instruction memory inside the simulator
7. Refresh GUI and displays

3: *Advance one step*: Once pressed it attempts the following (producing errors if any process faces an error)

1. Deal with any flushing or stalling that needs to happen, branch prediction, or jumps that will happen.
2. Push pipeline forward (according to step 1 results).
3. Update PC for next cycle (next press on "Advance one step"). In addition, fetch that instruction.
4. Do register reading for decode stage instructions
5. Do hazard detection
6. Forward values if any forwarding needs to happen due to read after write or use after load.
7. Execute instruction in execute stage.
    a. If it is a branch instruction, check result and update branch prediction table.
    b. Also, raise a flag if flushing needs to happen when predicted wrongly.
    c. Then update entry in branch prediction table.
8. Do memory operations of the instruction in the memory stage
9. Do the write back operations of the instruction in the write back stage
10. Refresh GUI and displays

4: *Reset*:

- Resets the pipeline by filling it with nops
- Resets the call stack
- Resets the PC to 0
- Remove all hazard messages
- Resets all registers and data memory to 0
- Refresh GUI and displays

5: *Assembly source*:

- Shows the parsed instructions in order (acts as view of the instructions in the instruction memory)

6: *Messages text box*:

- Displays errors during parsing.
- Displays the text of any exception thrown during the running of the simulator.

7: *Change base utility*:

- Changes the base of the addresses in all the views (Hexadecimal or decimal)

8: *Registers view:*

- Displays the current values of the 32 registers.

9: *Data memory view*:

- Displays the current values of the 16 locations in the data memory.

10: *Call stack view:*

- Displays the current instruction addresses in the stack

11: *PC view:*

- Displays current PC

12: *Pipeline view*:

- Displays each of the 5 instructions in the pipeline.
- The resources {Fetch, Decode, Execute, Memory, Write back} are displayed horizontally.
- Time is vertical
- When the pipeline display is full, it wraps around and overwrites the pipeline entry at the top.

13: *Hazard Messages View*:

- Displays the hazard messages from the hazard detection function of the simulator.
- Displays any forwarding that happened, if it happened.
- Displays any flushing or stalling for the next cycle, if required.

Input Specification

Rules for input to parser:

1. Instructions need to be end line separated.
2. Instruction name parsing is not case sensitive.
3. Register numbers are only from 0 to 31.
4. Register numbers need to be preceded by a dollar sign and no spaces between the dollar sign and the number
5. Arbitrary spaces and end lines are tolerated.
6. Commas should exist where they are expected to occur and would not be tolerated is missing or if they appear arbitrarily in other places.
7. Only the first N instructions corresponding to the N word size of the instruction memory is parsed.

Instructions syntax

Syntax of R-format (Excluding jr)

rformat $rd, $rs, $rt

Syntax of I-format (Including memory instructions but excluding branching instructions)

Iformat $rt, $rs, immediate

Syntax of branching instructions (ie. ble)

branch $rs, $rt, immediate

Syntax of Jump Register (jr)

jr $rs

Syntax of Jump Instructions (j, jal, or jmp)

jump immediate

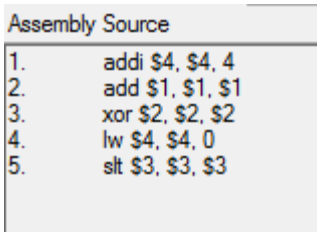
Syntax of Return (ret)

ret

Examples

1) Normal operation (No hazards)

Assembly Source Code:



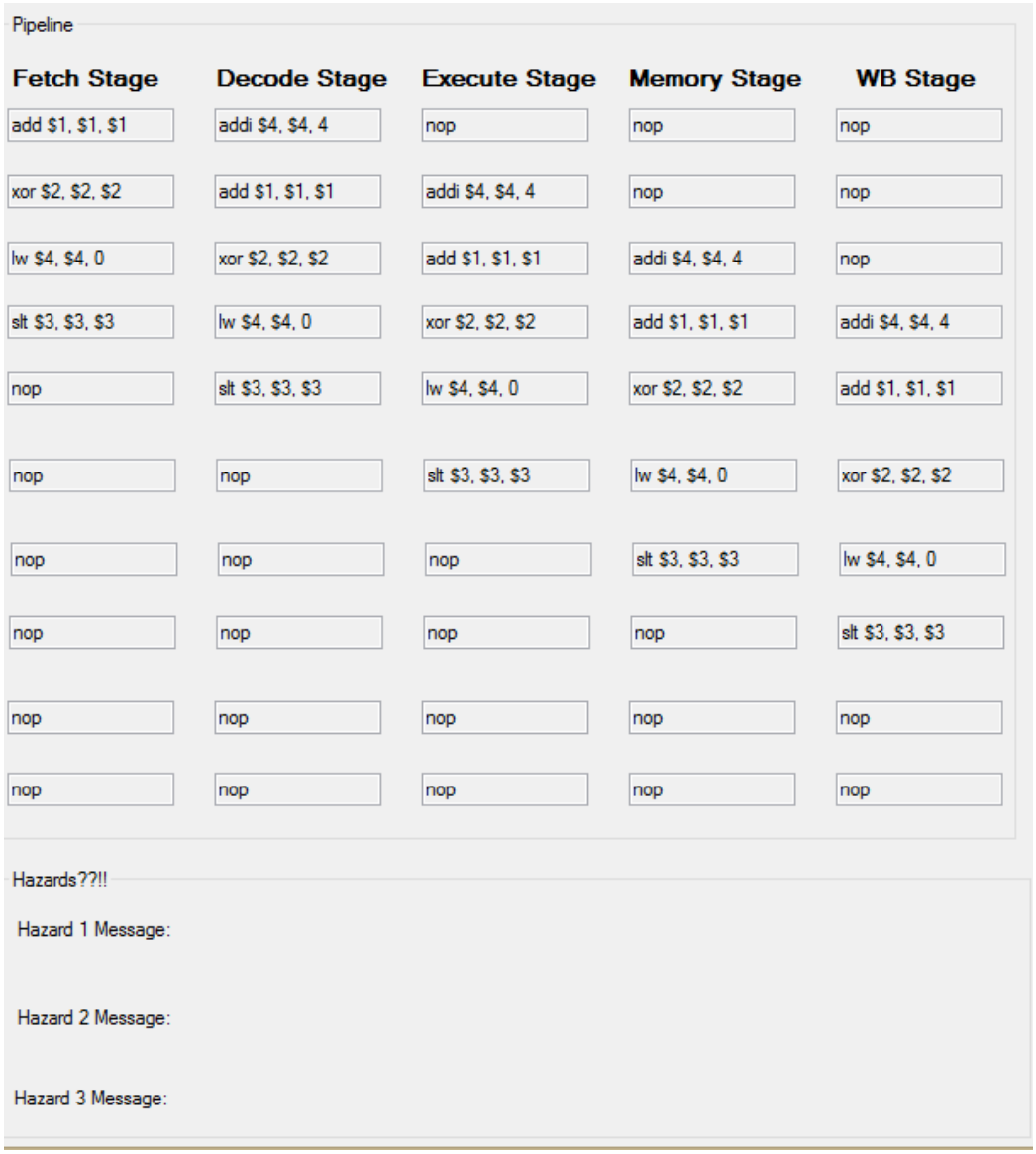


Figure 1.a: Pipeline

2) Read after write (Solved through forwarding)

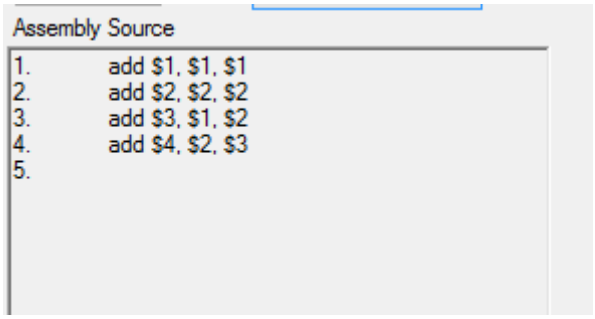


Figure 2.a: Assembly Source

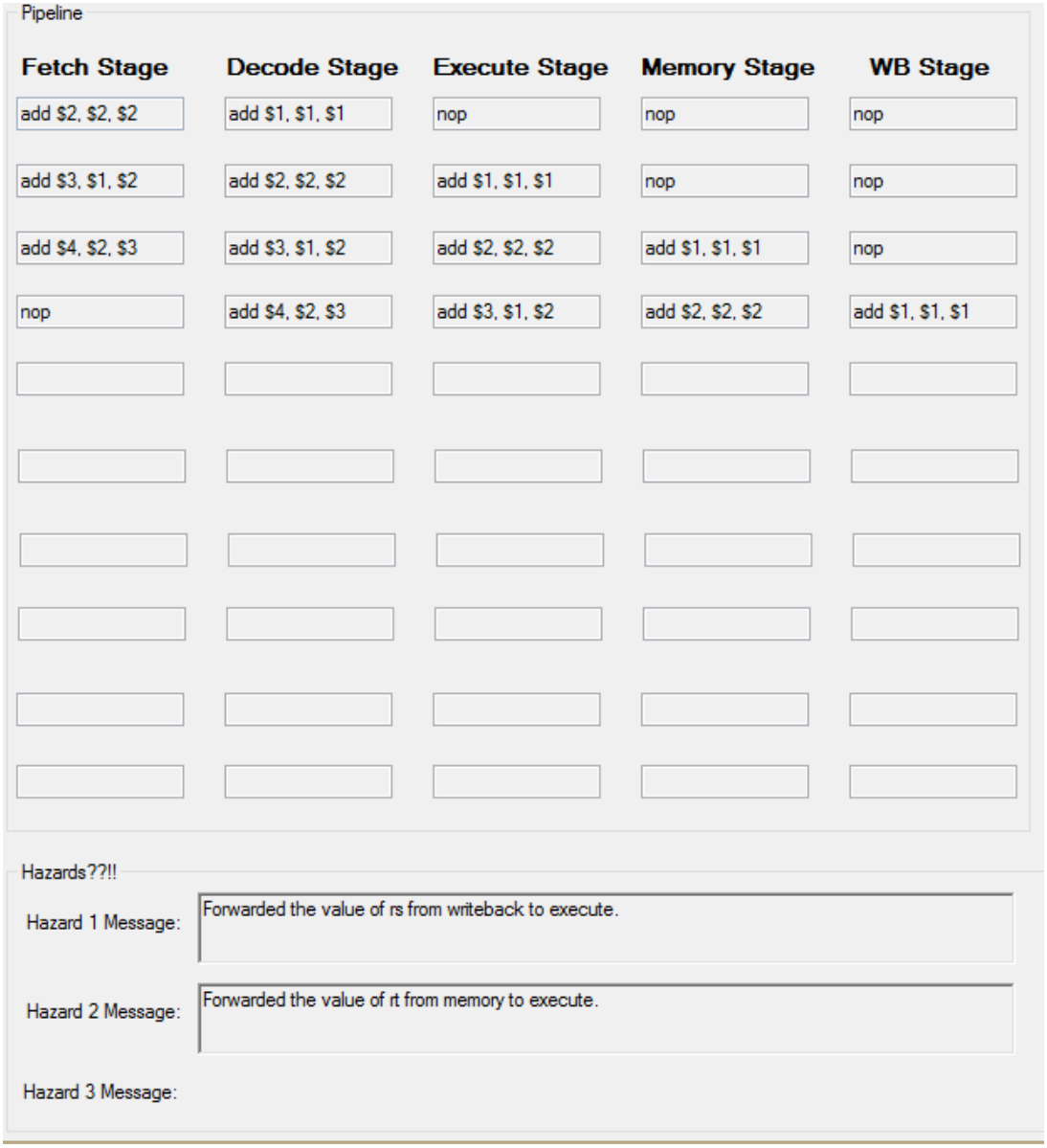


Figure 2.b: Pipeline and hazard messages

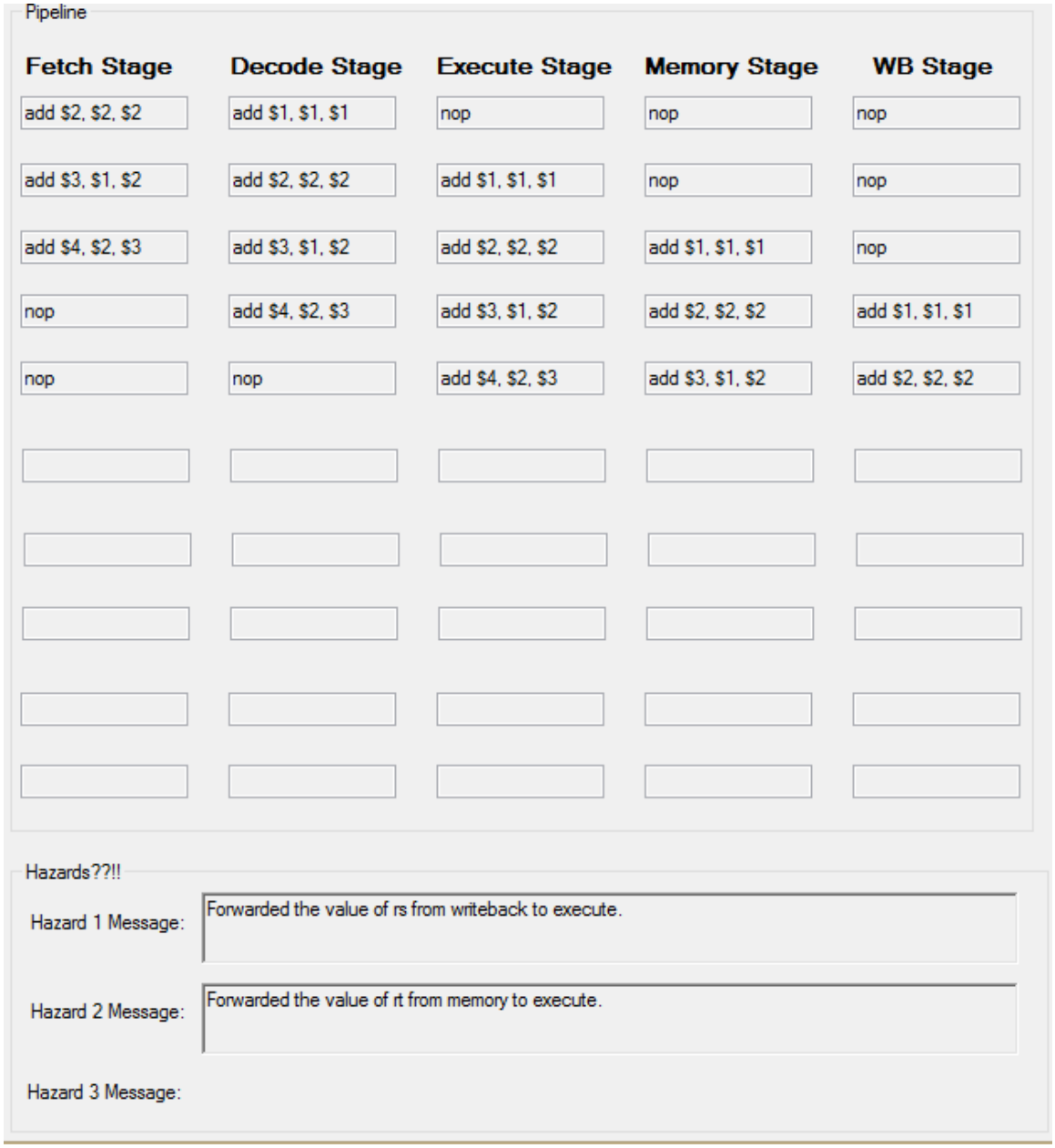


Figure 2.c: Pipeline and hazard messages

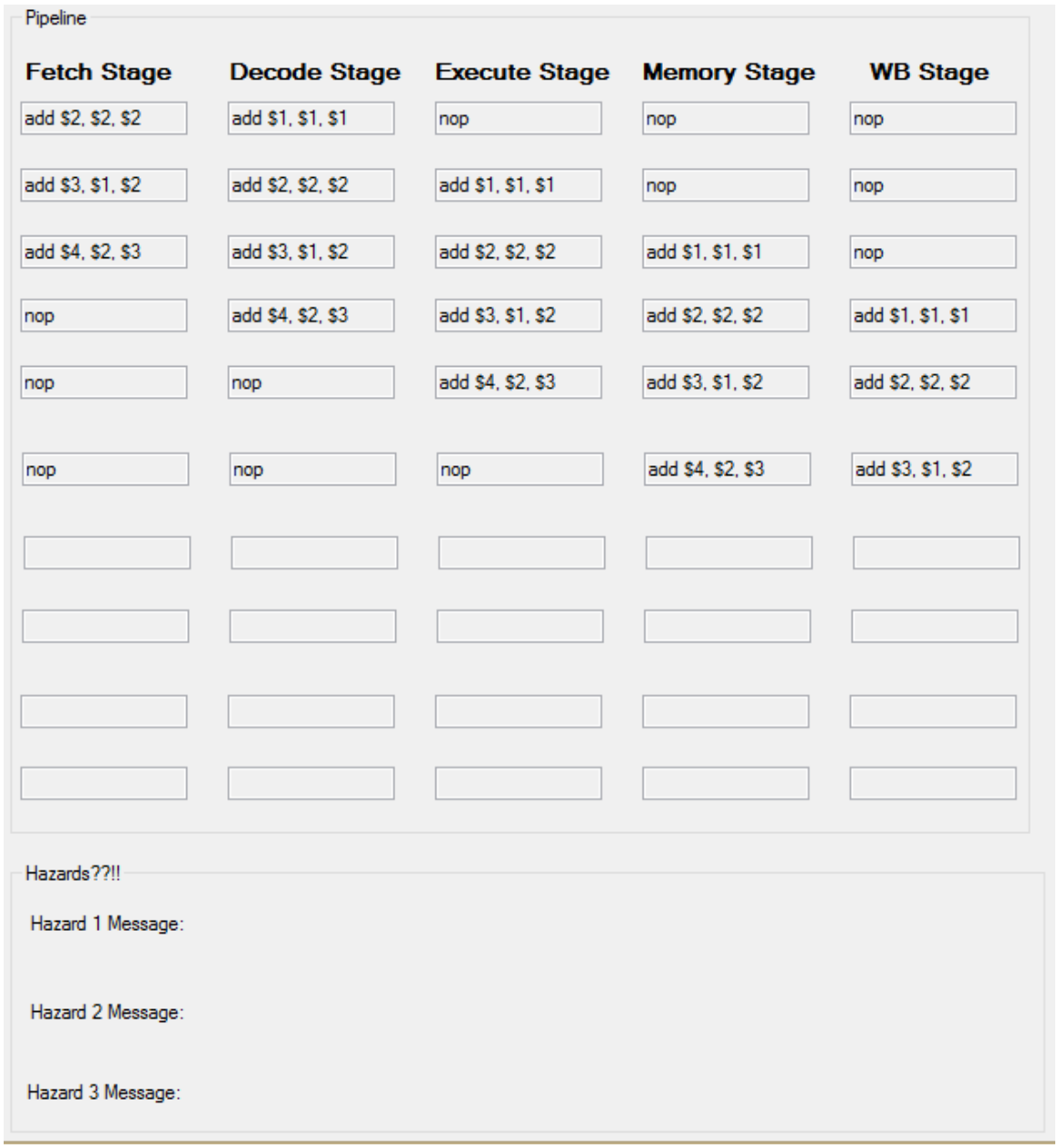


Figure 2.d: Pipeline and hazard messages

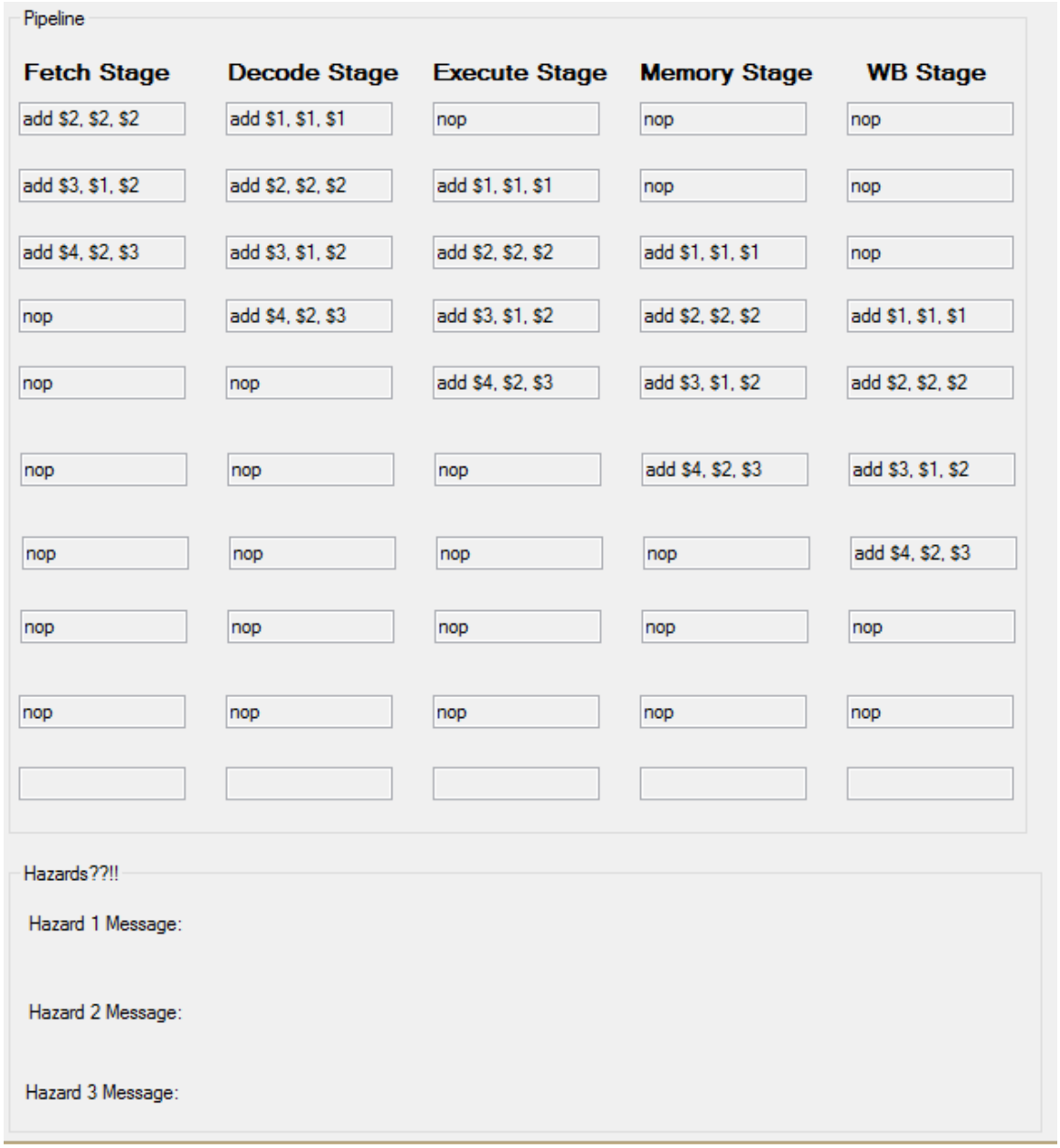


Figure 2.e: Pipeline and hazard messages

3) Read after load (Solved through stalling then forwarding)

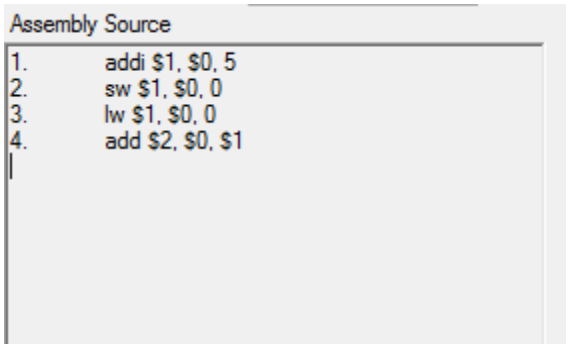


Figure 3.a: Assembly Source

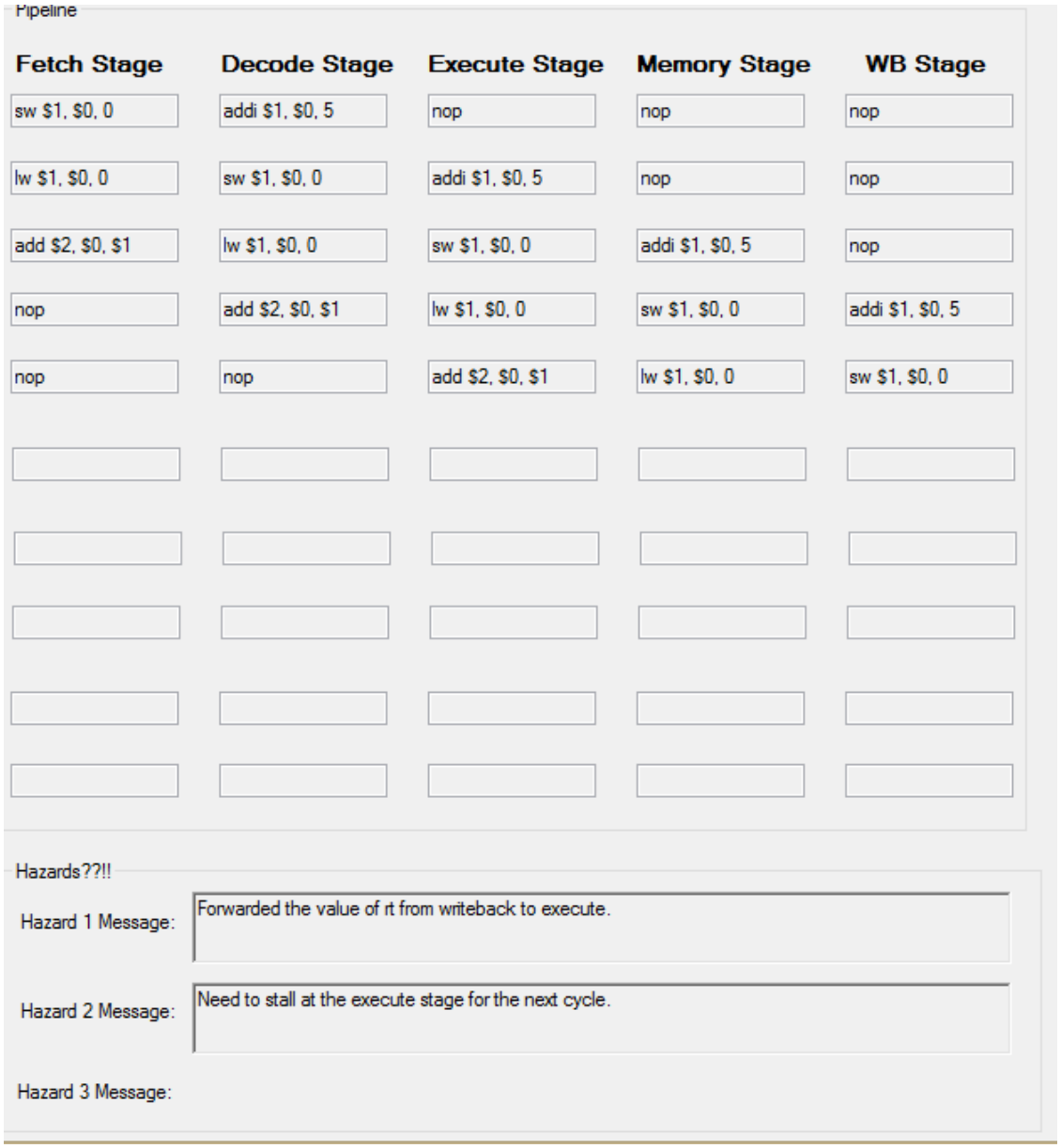


Figure 3.b: Pipeline and hazard messages

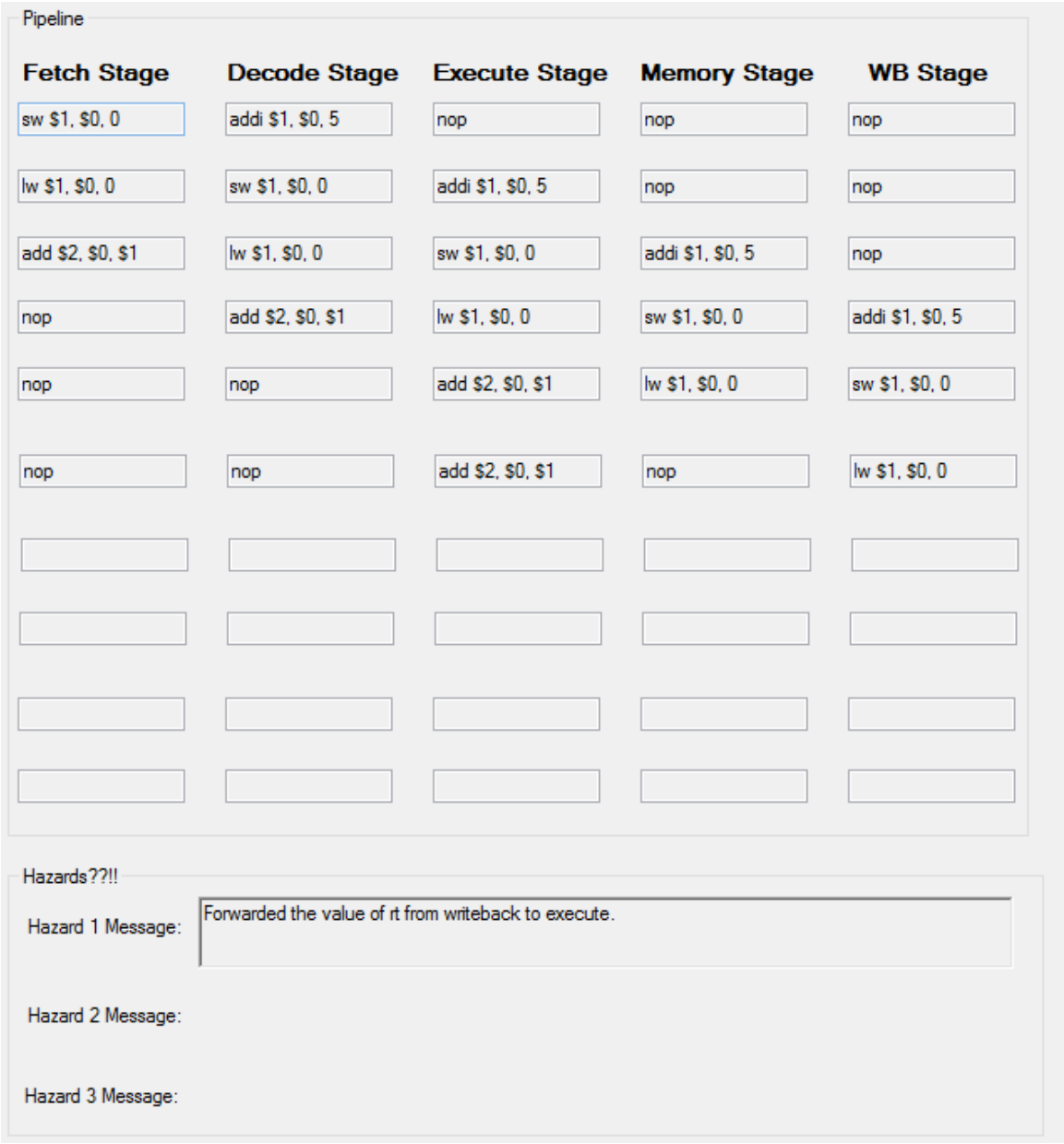


Figure 3.c: Pipeline and hazard messages

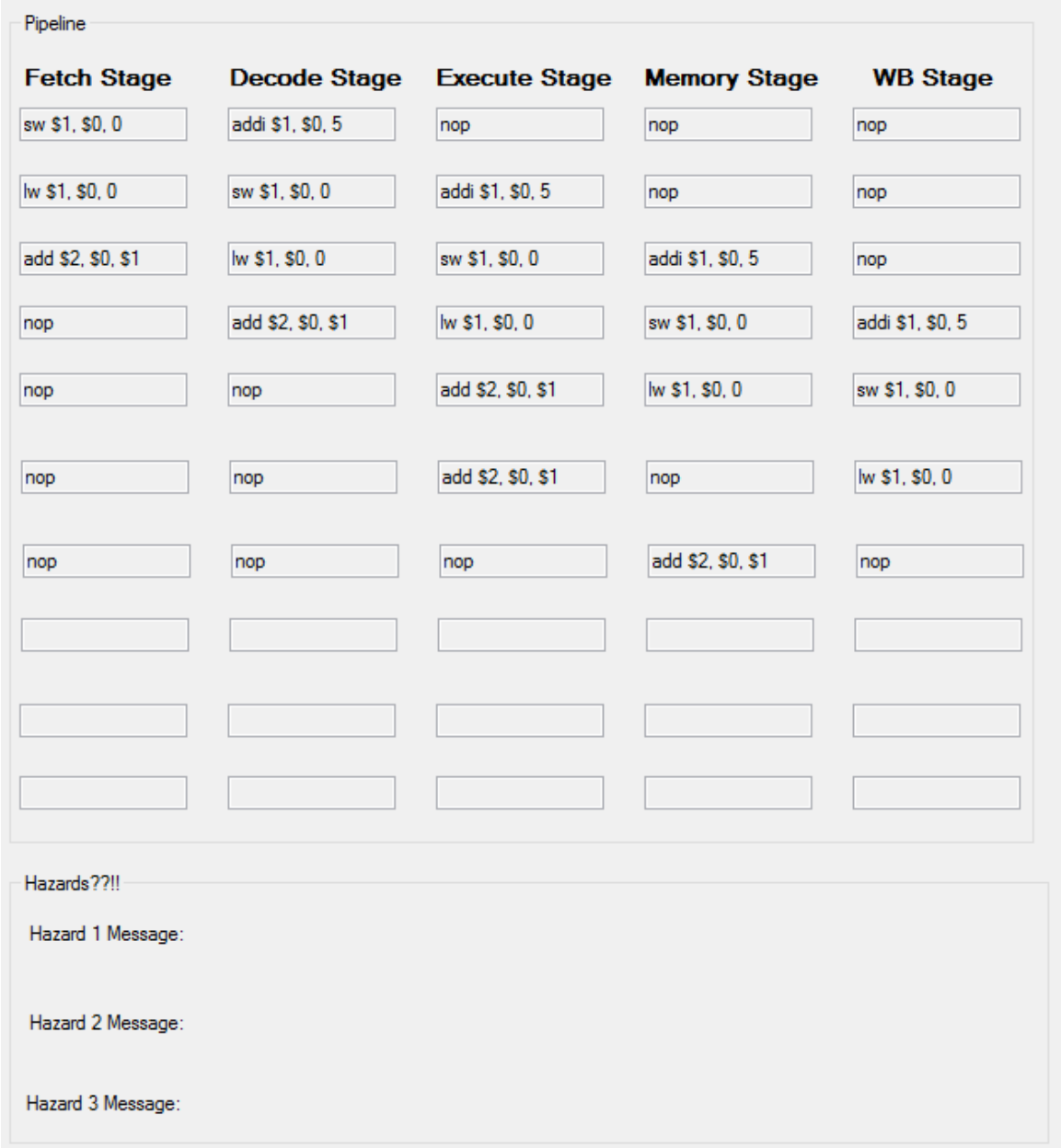


Figure 3.d: Pipeline and hazard messages

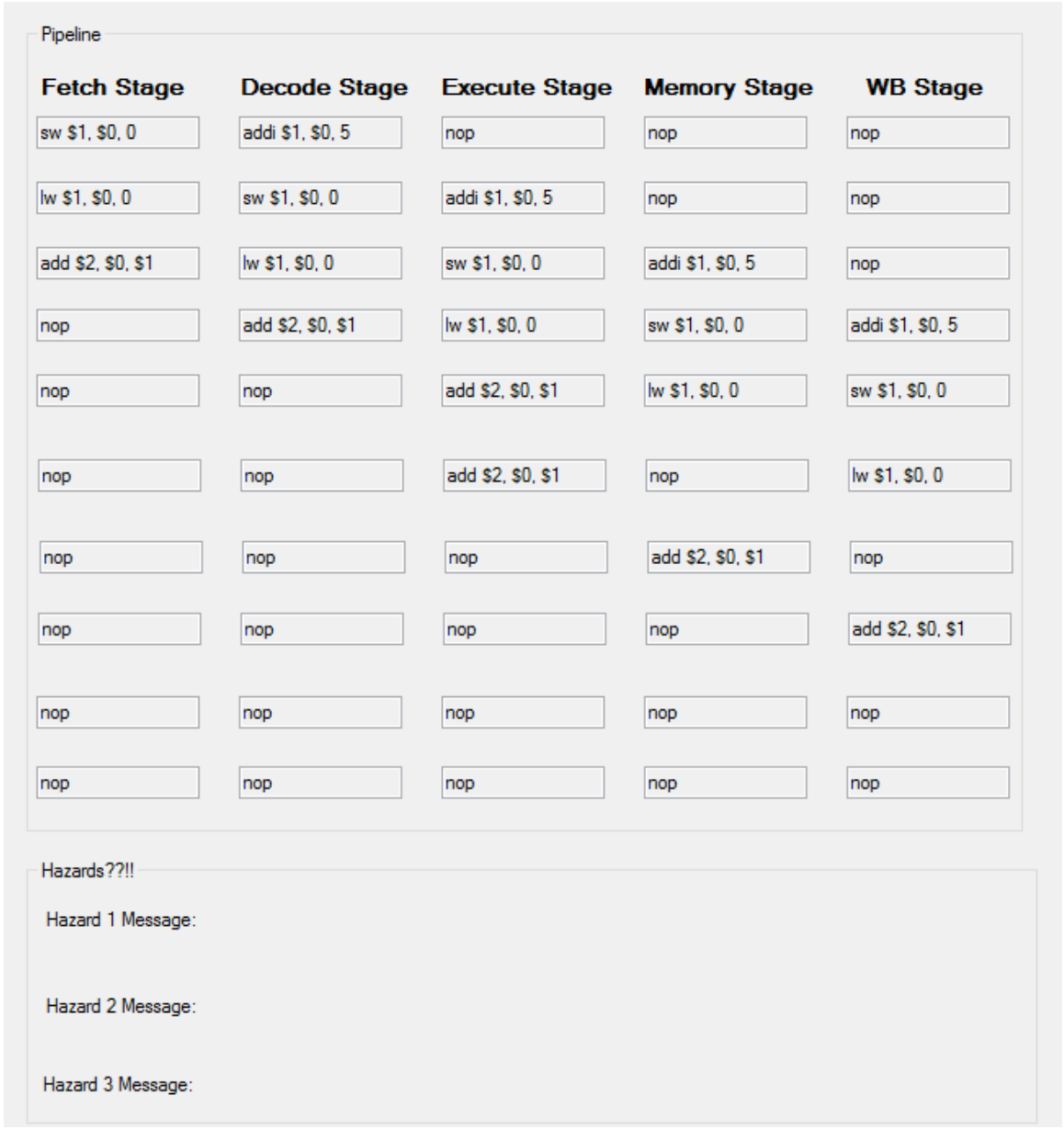


Figure 3.e: Pipeline and hazard messages

4) Fetched wrong instruction (solved through flushing)

Assembly source code:

Assembly Source

```
1.      j 2
2.      xor $2, $2, $2
3.      add $3, $1, $2
4.      j 0
5.      xor $2, $2, $2
```

Pipeline

Fetch Stage	Decode Stage	Execute Stage	Memory Stage	WB Stage
xor $2, $2, $2	j 2	nop	nop	nop

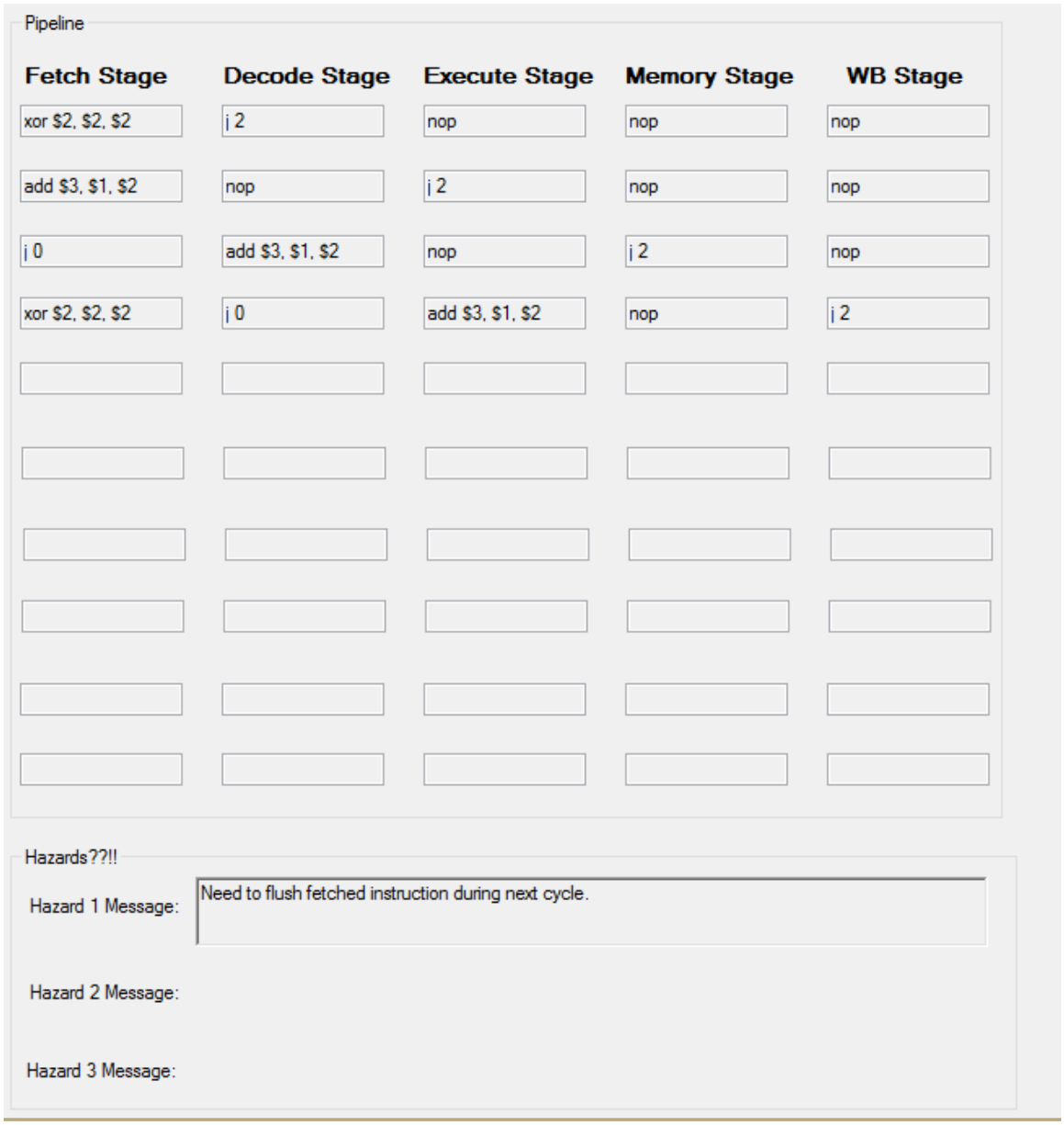
Hazards??!!

Hazard 1 Message: Need to flush fetched instruction during next cycle.

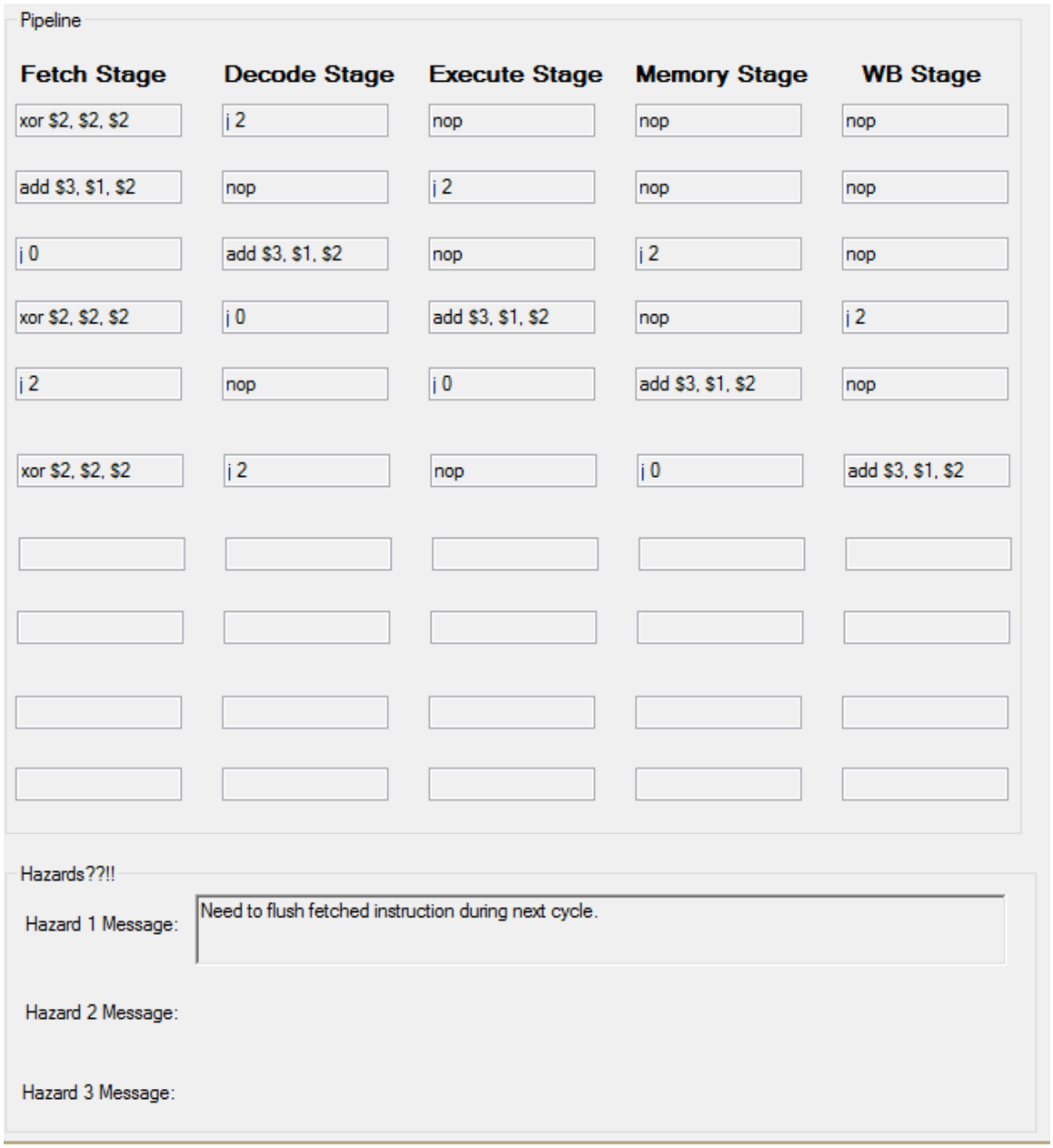
Hazard 2 Message:

Hazard 3 Message:

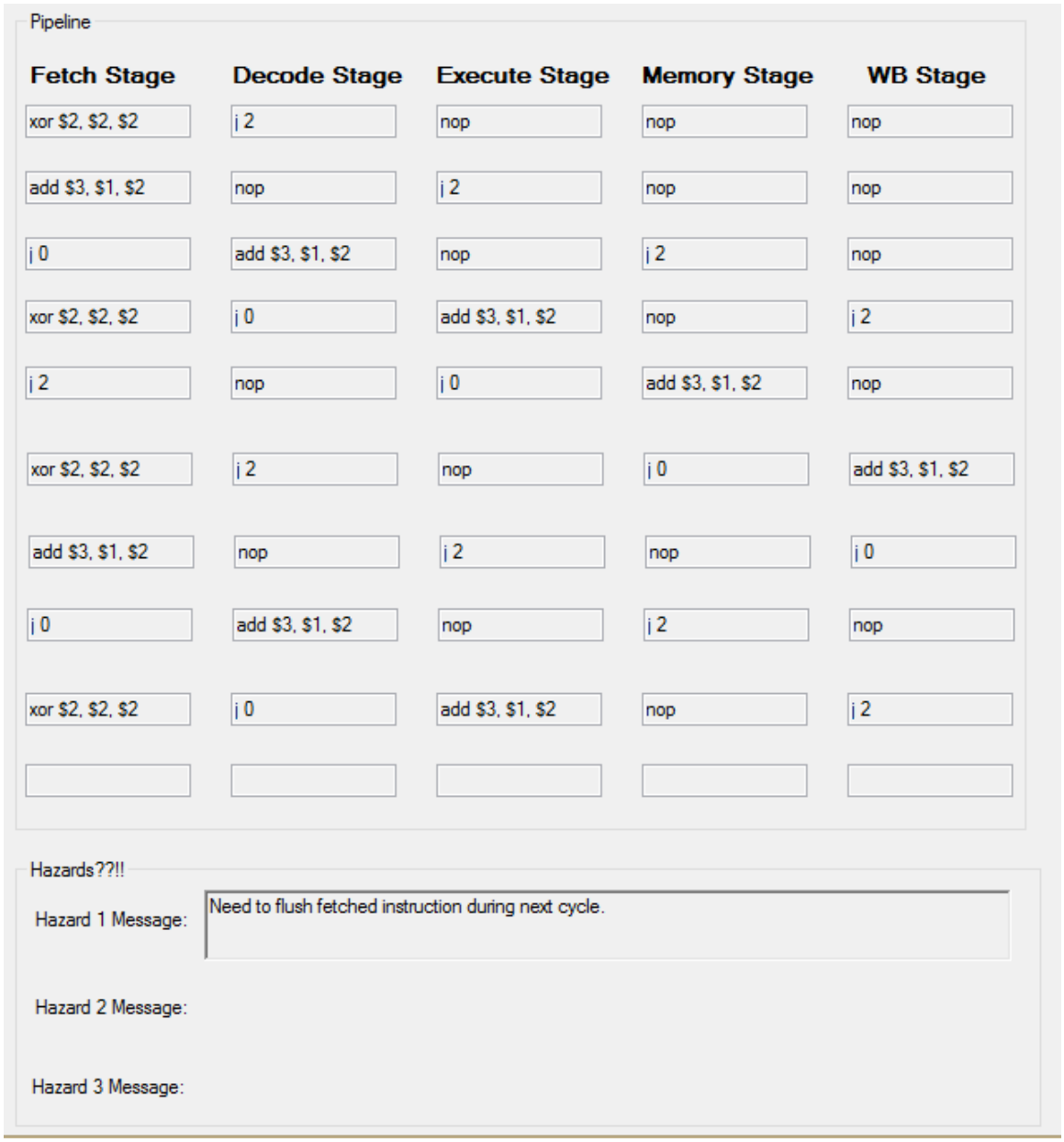
4.a Pipeline and hazard messages



4.b Pipeline and hazard messages



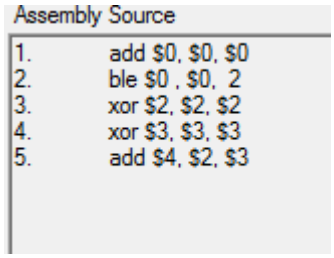
4.c Pipeline and hazard messages

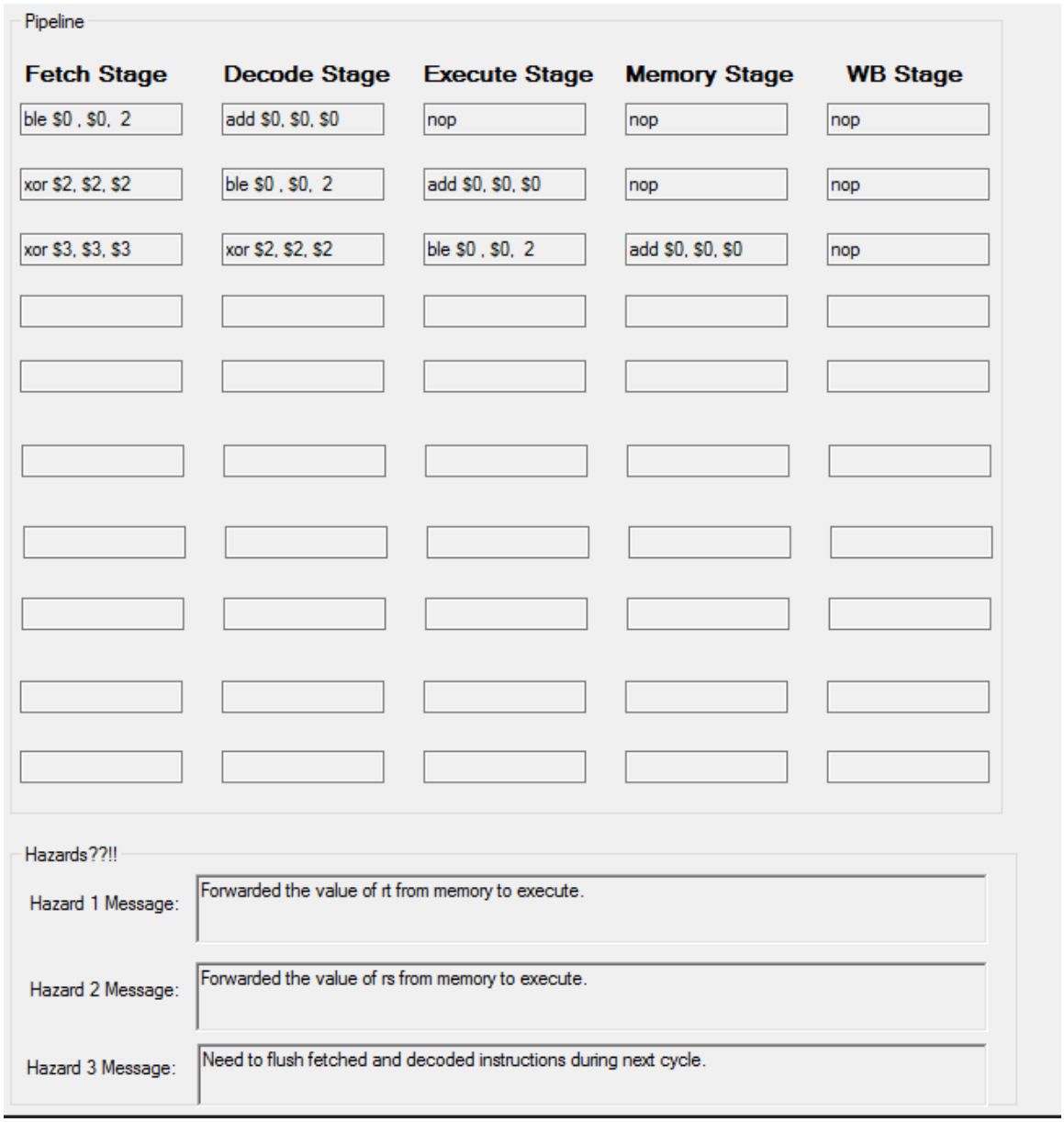


4.d Pipeline and hazard messages

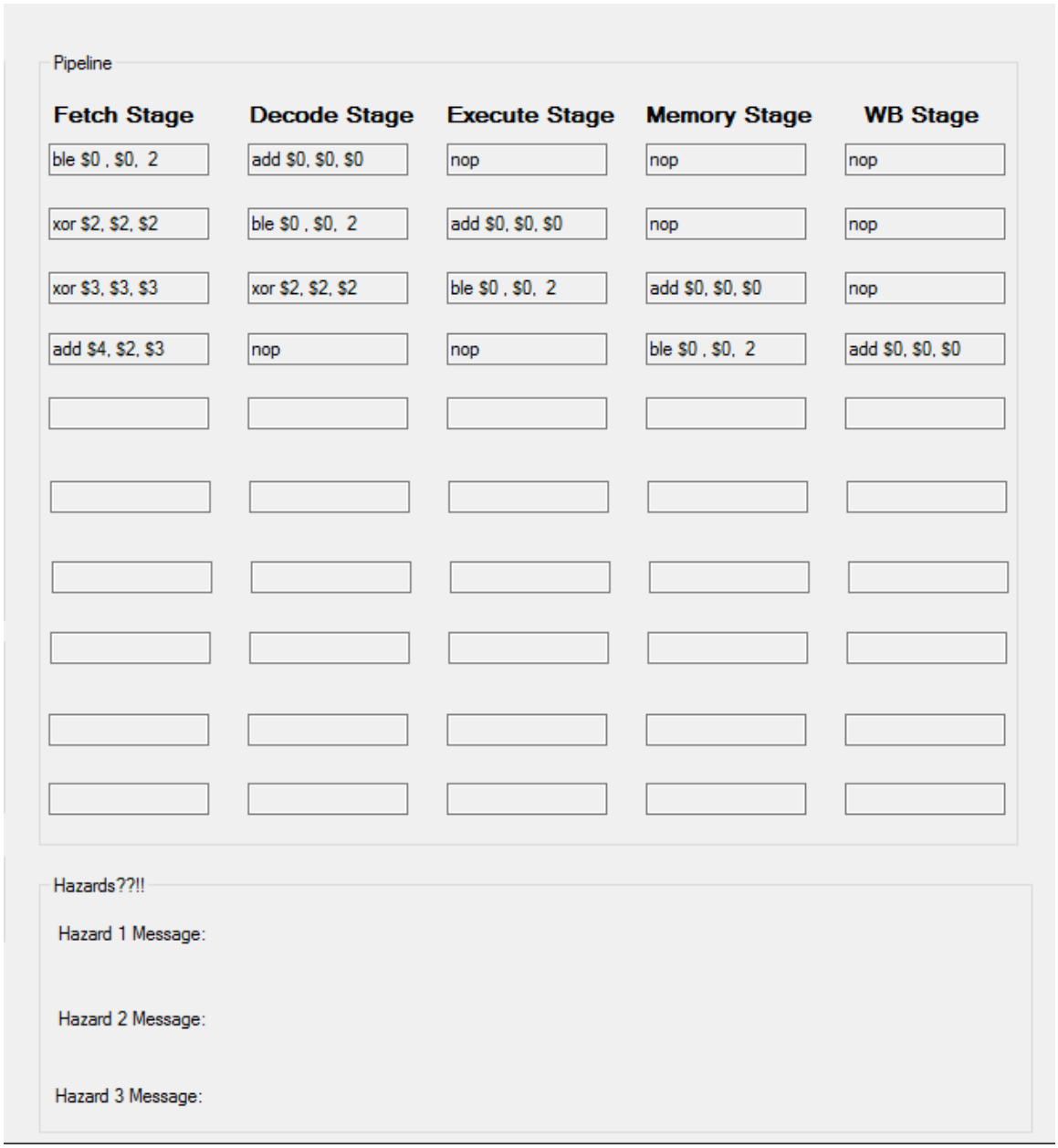
5) Fetched and decoded wrong instructions (solved through flushing)

Assembly source code:





5.a Pipeline and Hazards Messages



5.b Pipeline and Hazards Messages

6) Syntax Error

Assembly source code:

Assembly Source

```
1.      add $1. $2, $3
2.      ble $1, $2 , $400
3.      hello $6, $5 $4
```

Messages:

```
Syntax error: Cannot parse line 1
Syntax error: Cannot parse line 2
Syntax error: Cannot parse line 3
A fatal exception has occured using the parsing utility
Now resetting the environment
```

6.a Messages Box, showing errors

Appendix A (Code):

The application .exe and regex rule file are included in the submission folder.

Code is pasted below only for reading.

To access the code easily and without copy-paste errors **use the GitHub repo** @ **https://github.com/decltypeme/mips-simulator**

The code divided by filename.

main.cpp

```
#include <iostream>
#include <string>
#include "Simulator.h"
#include <iostream>
#define ___GUI_ENV
//#define ___CONSOLE_ENV
using namespace std;
using namespace System;
using namespace System::Windows::Forms;



void uiInitialize()
{
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

}


char* stageName[5] = { "Fetch", "Decode", "Execute", "Memory", "Write Back" };

void displayFetch()
{
	cout << endl << endl;
	cout << "PC:\t" << PC * 4;
	cout << endl << endl;
	cout << "Pipeline\t\t" << "Inst" << endl;
	cout << stageName[0] << "\t\t" << inst_memory[PC]->instString << endl;
	for (int i = 1; i < 5; ++i)
	{
		cout << stageName[i] << "\t\t" << pipeline[i - 1]->instString <<
endl;
	}
}

void displayHazards()
{
	cout << endl << endl;
	for (int i = 4; i >= 0; i--)
	{
		if (hazards[i])
		{
			hazardMsg* msgptr = gethazardMsgPtr(hazards[i]);
			cout << msgptr->hazard << endl;
		}
	}
}

void displayStorage()
{
	cout << endl << endl;
	cout << "RegNum\t" << "Value" << endl;
	for (int i = 0; i < 6; ++i)
	{
		cout << i << "\t" << registers[i] << endl;
```

```
	}
	cout << 31 << "\t" << registers[31] << endl;
	cout << endl << endl;
	cout << "MemNum\t" << "Value" << endl;
	for (int i = 0; i < 5; ++i)
	{
		cout << i << "\t" << data_memory[i] << endl;
	}
	cout << endl << endl;
}

void fillwithInst()
{
	inst_memory[0] = new Ble(0, 0, 0, 0, "Ble that branches to next inst");
	inst_memory[1] = new Xor(0, 0, 0, 4, "XOR $0, $0, $0");
}


#ifdef ___GUI_ENV
[STAThread]
#endif
int main()
{
#ifdef ___GUI_ENV
	initialize();
	uiInitialize();
	mips::Simulator mainSimulatorForm;
	Application::Run(%mainSimulatorForm);
	try {

	}
	catch (exception& ex)
	{
		string msg = ex.what();
		mainSimulatorForm.ParsingResults->AppendText(gcnew
String(msg.c_str()));
		mainSimulatorForm.ParsingResults->AppendText("\n");
	}
	catch (...)
	{
		mainSimulatorForm.ParsingResults->AppendText("Unhandled Exception");

	}
#endif
#ifdef ___CONSOLE_ENV
	initialize();

	fillwithInst();

	displayFetch();
	displayStorage();

	system("Pause");


	int n = 0;
	while (n<100)
	{
```

```
			n++;

			fetch();

			displayFetch();

			decode();
			execute();
			memory();
			writeBack();

			displayHazards();
			displayStorage();

			system("Pause");
		}

		cout << endl << endl << "End of simulation\n\n";
		system("Pause");
#endif
}
```

Utilities

1) utility.h

```
#pragma once
#include "datapath.h"
#include "hazards.h"
#include "Macro.h"
#include <string>

using namespace std;
class inst;

constexpr int maxRegNumber = 31;
constexpr int dataMemSize = 16;
constexpr int instMemSize = 16;

extern int PC;
extern int registers[maxRegNumber + 1];
extern int stack_size;
extern int stack[4];
extern int data_memory[dataMemSize];
extern inst* inst_memory[instMemSize];
extern inst* pipeline[4];
extern int hazards[5];
extern string hazard_msgs[3];

struct prediction
{
		int inst_address; bool taken;
		prediction()
			:inst_address(-1), taken(0) {}
		prediction(int _inst_address, bool _taken)
			:inst_address(_inst_address), taken(_taken) {}
};
extern prediction bpt[instMemSize];
```

```
extern hazardMsg* gethazardMsgPtr(int value);

bool validateRegister(const int& reg);

int readRegister(const int& reg);

void writeRegister(const int&reg, const int& value);

int readDataMem(const int& index);

void writeDataMem(const int& index, const int& value);

void fillNops();

void reset();

void initialize();

void pushtostack(int address);

int popfromstack();

int updatePC();

int predict_branch();

bool right_prediction();

prediction* getbptptr(prediction& totest, int inst_address);
```

2) utility.cpp

```
#include <cstring>
#include <stdexcept>
#include <string>
#include <algorithm>

#include "instructions.h"
using namespace std;


int PC = 0;
int registers[maxRegNumber + 1];
int stack_size = 0;
int stack[4];
int data_memory[dataMemSize];
inst* inst_memory[instMemSize];
inst* pipeline[4];
int hazards[5];
prediction bpt[instMemSize];
string hazard_msgs[3];

void reset()
{
```

```
	PC = 0;
	stack_size = 0;
	memset(registers, 0, sizeof(registers));
	memset(stack, 0, sizeof(stack));
	memset(data_memory, 0, sizeof(data_memory));
	memset(hazards, 0, sizeof(hazards));
	for (int i = 0; i < 4; ++i)
	{
		pipeline[i] = new inst();
	}
	for (int i = 0; i < instMemSize; ++i)
	{
		bpt[i] = prediction();
	}
}

hazardMsg* gethazardMsgPtr(int value)
{
	for (int i = 0; i < 14; i++)
	{
		if (hazardMsgs[i].value == value)
		{
			return &hazardMsgs[i];
		}
	}
	return nullptr;
}

bool validateRegister(const int& reg)
{
	return !(reg < 0 || reg >maxRegNumber);
}

int readRegister(const int& reg)
{
	if (!validateRegister(reg))
		throw invalid_argument("Invalid register passed");
	if (reg)
		return registers[reg];
	else
		return 0;
}

void writeRegister(const int&reg, const int& value)
{
	if (!validateRegister(reg))
		throw invalid_argument("Invalid register passed");
	if (reg)
		registers[reg] = value;
	return;
}

int readDataMem(const int& index)
{
	return data_memory[index];
}

void writeDataMem(const int& index, const int& value)
```

```
{
	data_memory[index] = value;
}

void fillNops()
{
	for (int i = 0; i < 4; ++i)
	{
		pipeline[i] = new inst();
	}

	for (int i = 0; i < instMemSize; ++i)
	{
		inst_memory[i] = new inst();
	}

	for (int i = 0; i < instMemSize; ++i)
	{
		bpt[i] = prediction();
	}
}

void initialize()
{
	PC = 0;
	stack_size = 0;
	memset(registers, 0, sizeof(registers));
	memset(stack, 0, sizeof(stack));
	memset(data_memory, 0, sizeof(data_memory));
	memset(hazards, 0, sizeof(hazards));
	fillNops();
}

void pushtostack(int address)
{
	if (stack_size < 4)
	{
		stack[stack_size++] = address;
	}
	else
		throw logic_error("Stack full.");
}

int popfromstack()
{
	if (stack_size > 0)
	{
		return stack[--stack_size];
	}
	else
		throw logic_error("Stack empty.");
}

int predict_branch()
{
	prediction* bptptr = nullptr;
	Ble* bleptr = dynamic_cast<Ble*> (pipeline[0]);
	for (int i = 0; i < instMemSize; ++i)
```

```
	{
		bptptr = getbptptr(bpt[i], bleptr->instAddress);
		if (bptptr)
			break;
	}

	if (bptptr)
	{
		if (bptptr->taken)
		{
			if (bleptr->addressIfTaken != PC)
			{
				return bleptr->addressIfTaken;
			}
			else
			{
				return PC + 1;
			}
		}
		else
		{
			if (bleptr->addressIfNotTaken != PC)
			{
				return bleptr->addressIfNotTaken;
			}
			else
			{
				return PC + 1;
			}
		}
	}
	else
	{
		bpt[static_cast<unsigned int>(bleptr->instAddress) % instMemSize] =
prediction(bleptr->instAddress, 0);
		if (bleptr->addressIfNotTaken != PC)
		{
			return bleptr->addressIfNotTaken;
		}
		else
		{
			return PC + 1;
		}
	}
}

prediction* getbptptr(prediction& totest, int inst_address)
{
	if (totest.inst_address == inst_address) return &totest;
	else return nullptr;
}

bool right_prediction()
{
	prediction* bptptr = nullptr;
	Ble* bleptr = dynamic_cast<Ble*> (pipeline[1]);
	for (int i = 0; i < instMemSize; ++i)
	{
```

```
			bptptr = getbptptr(bpt[i], bleptr->instAddress);
			if (bptptr)
				break;
		}

		if (bptptr)
		{
			int branchedToLastTime;
			if (bptptr->taken)
			{
				branchedToLastTime = bleptr->addressIfTaken;
			}
			else
			{
				branchedToLastTime = bleptr->addressIfNotTaken;
			}

			if (branchedToLastTime != bleptr->addressTrue)
			{
				bptptr->taken = !(bptptr->taken);
				bleptr->wrongPredicitonFlag = 1;
				return false;
			}
			else
			{
				bleptr->wrongPredicitonFlag = 0;
				return true;
			}
		}
		else
			throw logic_error("No prediction record of this branching
instruction");
}

int updatePC()
{
		J* jptr = dynamic_cast <J*> (pipeline[0]);
		Jr* jrptr = dynamic_cast <Jr*> (pipeline[0]);
		Ret* retptr = dynamic_cast <Ret*> (pipeline[0]);
		Ble* bleptr0 = dynamic_cast <Ble*> (pipeline[0]);
		Ble* bleptr1 = dynamic_cast <Ble*> (pipeline[1]);

		if (bleptr1 != nullptr)
		{
			if (bleptr1->wrongPredicitonFlag == 0)
			{
				if (jptr)
					if (jptr->address != PC)
					{
						return jptr->address;
					}
					else
					{
						return PC + 1;
					}
				if (retptr)
				{
					if (retptr->addressPopped != PC)
```

```
                        {
                                return retptr->addressPopped;
                        }
                        else
                        {
                                return PC + 1;
                        }
                }
                if (bleptr0)
                {
                        return predict_branch();
                }
                if (jrptr)
                {
                        if (jrptr->rsData != PC)
                        {
                                return jrptr->rsData;
                        }
                        else
                        {
                                return PC + 1;
                        }
                }

                return PC + 1;
        }
        else
        {
                bleptr1->wrongPredicitonFlag = 0;
                if (bleptr1->addressTrue != PC)
                {
                        return bleptr1->addressTrue;
                }
                else
                {
                        return PC + 1;
                }
        }
}
else
{
        if (jptr)
                if (jptr->address != PC)
                {
                        return jptr->address;
                }
                else
                {
                        return PC + 1;
                }
        if (retptr)
        {
                if (retptr->addressPopped != PC)
                {
                        return retptr->addressPopped;
                }
                else
                {
```

```
                        return PC + 1;
                }
        }
        if (bleptr0)
        {
                return predict_branch();
        }
        if (jrptr)
        {
                if (jrptr->rsData != PC)
                {
                        return jrptr->rsData;
                }
                else
                {
                        return PC + 1;
                }
        }

        return PC + 1;
    }
}
```

3) Macro.h

```
#pragma once

#define ___updateStackField(___StackField)\
                if (stack_size > ___StackField)\
                        {\
                                Stack##___StackField ->Text = gcnew
System::String(fromContentToString(stack[##___StackField]).c_str());\
                        }\
                else\
                        {\
                                Stack##___StackField ->Text = gcnew
System::String("--------");\
                        }

/*#define ___stackCounter 4

#define ___updateStackFieldLoop\
\#if ___stackCounter > 0\
___updateStackField(___stackCounter)\
\#define ___stackCounter (___stackCounter - 1)\
___updateStackFieldLoop\
\#endif*/

#define ____THROW_IN_FORM(___REST_OF_LINE)\
    try {\
___REST_OF_LINE\
    }\
    catch (exception& ex)\
    {\
        string msg = ex.what();\
        ParsingResults->AppendText(gcnew String(msg.c_str()));\
```

```
		ParsingResults->AppendText("\n");\
	}\
	catch (...)\
	{\
		ParsingResults->AppendText("Unhandled Exception");\
	}

#define ___CATCH_IN_FORM()\
catch (exception& ex)\
{\
string msg = ex.what(); \
String^ temp = gcnew String(msg.c_str());\
ParsingResults->AppendText(temp); \
ParsingResults->AppendText("\nNow resetting the environment\n"); \
resetTheSimEnv();\
}\
catch (...)\
{\
ParsingResults->AppendText("Unhandled Exception"); \
ParsingResults->AppendText("\n Now resetting the environment\n"); \
resetTheSimEnv();\
}

#define ___print_Inst_TextBox(___TIME_ID)\
Inst0p##___TIME_ID ->Text = gcnew String(pipeline[0]->instString.c_str());\
Inst1p##___TIME_ID ->Text = gcnew String(pipeline[1]->instString.c_str());\
Inst2p##___TIME_ID ->Text = gcnew String(pipeline[2]->instString.c_str());\
Inst3p##___TIME_ID ->Text = gcnew String(pipeline[3]->instString.c_str());

#define ___SetColorBlue(___TIME_ID)\
Inst0p##___TIME_ID ->ForeColor = System::Drawing::Color::Red;\
Inst1p##___TIME_ID ->ForeColor = System::Drawing::Color::Red;\
Inst2p##___TIME_ID ->ForeColor = System::Drawing::Color::Red;\
Inst3p##___TIME_ID ->ForeColor = System::Drawing::Color::Red;

#define ___SetColorBlack(___TIME_ID)\
Inst0p##___TIME_ID ->ForeColor = System::Drawing::Color::Blue;\
Inst1p##___TIME_ID ->ForeColor = System::Drawing::Color::Blue;\
Inst2p##___TIME_ID ->ForeColor = System::Drawing::Color::Blue;\
Inst3p##___TIME_ID ->ForeColor = System::Drawing::Color::Blue;



#define ___printFetchBox(___TIME_ID)\
InstFp##___TIME_ID ->Text = gcnew String(inst_memory[PC]->instString.c_str());

#define ___resetAllPipeline(___TIME_ID)\
Inst0p##___TIME_ID ->Text = gcnew String("");\
Inst1p##___TIME_ID ->Text = gcnew String("");\
Inst2p##___TIME_ID ->Text = gcnew String("");\
Inst3p##___TIME_ID ->Text = gcnew String("");\
InstFp##___TIME_ID ->Text = gcnew String("");


#define ___HAZARD_DISPLAY(___HAZARD_MSG_ID)\
if(!hazard_msgs[##___HAZARD_MSG_ID - 1].empty())\
{\
BoxH##___HAZARD_MSG_ID ->Visible = true;\
```

```
BoxH##___HAZARD_MSG_ID ->Visible = true;\
BoxH##___HAZARD_MSG_ID ->Text = gcnew String(hazard_msgs[##___HAZARD_MSG_ID -
1].c_str());\
}\
else\
{\
BoxH##___HAZARD_MSG_ID ->Visible = false;\
BoxH##___HAZARD_MSG_ID ->Visible = false;\
BoxH##___HAZARD_MSG_ID ->Text = "";\
}
```

Parsing Related

1) regexRule.txt

```
([Aa][Dd][Dd]|[Xx][Oo][Rr]|[Ss][Ll][Tt])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-
9]{1}|[0-9])\s*,\s*\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*,\s*\$(1[0-
9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*(?:\#(?:.*)){0,1}
([Aa][Dd][Dd][Ii]|[Ll][Ww]|[Ss][Ww]|[Bb][Ll][Ee])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-
1]{1}|0[0-9]{1}|[0-9])\s*,\s*\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-
9])\s*,\s*(0x[0-9A-Fa-f]+|(?:0d){0,1}[0-9\-]+)\s*(?:\#(?:.*)){0,1}
([Jj][Aa][Ll]|[Jj]|[Jj][Mm][Pp])\s+(0x[0-9A-Fa-f]+|(?:0d){0,1}[0-9\-
]+)\s*(?:\#(?:.*)){0,1}
([Jj][Rr])\s+\$(1[0-9]{1}|2[0-9]{1}|3[0-1]{1}|0[0-9]{1}|[0-9])\s*(?:\#(?:.*)){0,1}
([Rr][Ee][Tt])\s*(?:\#(?:.*)){0,1}
\s*(?:\#(?:.*)){0,1}
```

2) parser.h

```
#pragma once
#include "instructions.h"
#include <regex>
#include <sstream>
#include <string>
#include <ios>
#include <iomanip>
#include <algorithm>
typedef unsigned int immediateType;

using namespace std;

enum class instGenericType
{
	rType,
	iType,
	jType,
	jrType,
	retType,
	noOps,
	numTypes
};
constexpr int instTypeCount = int(instGenericType::numTypes);

//A function that resolves the immediate field
immediateType resolveImmediate(const string& strImmediate, bool signExtend =
true);
//A function that resolves a jump immediate
immediateType resolveJImmediate(const string& strImmediate);
```

```
/*
Verify that a given instruction is syntactically correct
*/
bool verifyInstruction(const string&instString, const vector<regex>& instRules);
/*
Instruction rules need to be passed with same order of appearance in the enum
class
Throws an exception
*/
inst* parseInstruction(const string&instString, const vector<regex>& instRules,
int address);

int fileHandler(int argc, char** argv, inst** instsToFill,
System::Windows::Forms::RichTextBox^ sourceField,
System::Windows::Forms::RichTextBox^ resultsField);
```

3) parser.cpp

```
#include "parser.h"
regex hex_reg = regex("(0x)([0-9A-Fa-f]+)");
regex dec_reg = regex("(0d){0,1}([0-9\-]+)");
immediateType resolveImmediate(const string& strImmediate, bool signExtend)
{
	smatch results;
	unsigned __int16 imm16;
	if (regex_match(strImmediate, hex_reg))
	{
		regex_search(strImmediate, results, hex_reg);
		stringstream ss(results[2]);
		ss >> hex >> imm16;
	}
	else if (regex_match(strImmediate, dec_reg))
	{
		regex_search(strImmediate, results, dec_reg);
		stringstream ss(results[2]);
		ss >> imm16;
	}
	else
	{
		throw invalid_argument("Invalid immediate: Unable to parse the
immediate field");
	}
	if (signExtend)
	{
		int value = (0x0000FFFF & imm16);
		int mask = 0x00008000;
		if (mask & imm16) {
			value += 0xFFFF0000;
		}
		return value;
	}
	else
	{
		return imm16;
	}
}
```

```
immediateType resolveJImmediate(const string& strImmediate)
{
	smatch results;
	int jAddr;
	if (regex_match(strImmediate, hex_reg))
	{
		regex_search(strImmediate, results, hex_reg);
		stringstream ss(results[2]);
		ss >> hex >> jAddr;
	}
	else if (regex_match(strImmediate, dec_reg))
	{
		regex_search(strImmediate, results, dec_reg);
		stringstream ss(results[2]);
		ss >> jAddr;
	}
	else
	{
		throw invalid_argument("Invalid immediate: Unable to parse the
immediate field");
	}
	return jAddr;
}
inst* parseInstruction(const string& instString, const vector<regex>& instRules,
int address)
{
	if (!verifyInstruction(instString, instRules))
		throw invalid_argument("Cannot parse an invalid instruction");
	smatch params;
	for (const auto& instRule : instRules)
	{
		regex_search(instString, params, instRule);
		if (!params.empty())
		{
			break;
		}
	}
	//Now, we will do a must-do cases, I know, you know it sucks, even this
plastic debugging duck knows it!
	//transform(params[1].begin(), params[1].end(), params[1].begin(),
toupper);
	string instName = params[1];
	for (int ii_it = 0; ii_it < instName.size(); ii_it++)
	{
		instName[ii_it] = toupper(instName[ii_it]);
	}
	//R-Type instructions
	if (instName.compare(string("ADD")) == 0 || instName == "XOR" || instName
== "SLT")
	{
		int rs = stoi(params[3]);
		int rt = stoi(params[4]);
		int rd = stoi(params[2]);
		if (instName.compare(string("ADD")) == 0)
		{
			return new Add(rd, rs, rt, address, instString);
		}
		else if (instName == "XOR")
```

```
                {
                        return new Xor(rd, rs, rt, address, instString);
                }
                else if (instName == "SLT")
                {
                        return new Slt(rd, rs, rt, address, instString);
                }
        }
        //I-Type Instructions

        else if (instName == "ADDI" | instName == "LW" | instName == "SW" |
instName == "BLE")
        {
                int rt = stoi(params[2]);
                int rs = stoi(params[3]);
                immediateType immediate = resolveImmediate(params[4]);
                if (instName == "ADDI")
                {
                        return new Addi(rt, rs, immediate, address, instString);
                }
                if (instName == "LW")
                {
                        return new Lw(rt, rs, immediate, address, instString);
                }
                else if (instName == "SW")
                {
                        return new Sw(rt, rs, immediate, address, instString);
                }
                else if (instName == "BLE")
                {
                        return new Ble(rs, rt, immediate, address, instString);
                }
        }
        else if (instName == "JR")
        {
                return new Jr(stoi(params[2]), address, instString);
        }
        else if (instName == "J" | instName == "JAL" || instName == "JMP")
        {
                immediateType jAddr = resolveJImmediate(params[2]);
                if (instName == "J")
                {
                        return new J(jAddr, address, instString);
                }
                else if (instName == "JAL")
                {
                        return new Jal(jAddr, address, instString);
                }
                else if (instName == "JMP")
                {
                        return new Jmp(jAddr, address, instString);
                }
        }
        else if (instName == "RET")
        {
                return new Ret(address, instString);
        }
        else
```

```
	{
		throw logic_error("Unrecognized Instruction: Instruction was
unrecognized as part of the ISA: " + to_string(address));
	}
	throw logic_error("Reached end of function, no suitable return");
}
bool verifyInstruction(const string& instString, const vector<regex>& instRules)
{
	for (const auto& rule : instRules)
	{
		if (regex_match(instString, rule))
			return true;
	}
	return false;
}
```

4) fileHandling.cpp

```
#include<iostream>
#include<regex>
#include<string>
#include<vector>
#include<fstream>
#include<stdexcept>

#include "parser.h"

using namespace std;


void readFromFileIntoVector(vector<string>& vectorToPush, string fileName, char
delimeter)
{
	ifstream ss(fileName);
	if (ss.fail())
		throw invalid_argument("File failed to open");
	while (!ss.eof())
	{
		string temp;
		getline(ss, temp, delimeter);
		vectorToPush.push_back(temp);
	}
	ss.close();
}

void readFromFileIntoPairVector(vector<pair<string, string>>& vectorToPush, string
fileName, char delimeter)
{
	ifstream ss(fileName);
	if (ss.fail())
		throw invalid_argument("File failed to open");
	while (!ss.eof())
	{
		string temp, s1, s2;
		ss >> s1 >> s2;
		getline(ss, temp, delimeter);
```

```
			vectorToPush.push_back(make_pair(s1, s2));
		}
		ss.close();
}

bool verifyLines(const vector<string>& ruleStrings, const vector<string> lines,
vector<int>& errorLines)
{
		bool toReturn = true;
		vector<regex> rules;
		for (auto& ruleString : ruleStrings)
		{
			rules.push_back(regex(ruleString));
		}
		for (int ii = 1; ii <= int(lines.size()); ii++)
		{
			const string& line = lines[ii - 1];
			bool isValid = false;
			for (auto& rule : rules)
			{
				if (regex_match(line, rule))
				{
					isValid = true;
					break;
				}
			}
			if (!isValid) {
				toReturn = false;
				errorLines.push_back(ii);
			}

		}
		return toReturn;
}

//It appends the instructions when parsing, make sure it is clear.
int fileHandler(int argc, char** argv, inst** instsToFill,
System::Windows::Forms::RichTextBox^ sourceField,
System::Windows::Forms::RichTextBox^ resultsField)
{
		if (argc == 4)
		{
			try {
				int counterOnEarth = 0;
				vector<string> linesToVerify;
				vector<string> str_rules;
				vector<int> errorLines;
				readFromFileIntoVector(linesToVerify, argv[3], '\n');
				readFromFileIntoVector(str_rules, argv[2], '\n');
				vector<regex> reg_rules;
				for (auto& ruleString : str_rules)
				{
					reg_rules.push_back(regex(ruleString));
				}
				if (string(argv[1]) == string("-verify"))
				{
					int lineCount = 1;
					for (const string& toParse : linesToVerify)
```

```
                        {
                                sourceField->AppendText(gcnew
System::String((to_string(lineCount) + ".\t" + toParse).c_str()));
                                sourceField->AppendText("\n");
                                lineCount++;
                        }
                        if (verifyLines(str_rules, linesToVerify, errorLines))
                        {
                                resultsField->AppendText("The file is
syntactically correct\n");
                                cout << "The file is syntactically correct" <<
endl;
                        }
                        else
                        {
                                for (auto& no : errorLines)
                                {
                                        resultsField->AppendText(gcnew
System::String(("Syntax error: Cannot parse line " + to_string(no) +
"\n").c_str()));
                                        cerr << "Syntax error: Cannot parse line
" << no << endl;
                                }
                        }

                }
                else if ((string(argv[1]) == string("-parse")))
                {
                        for (const auto& toParse : linesToVerify)
                        {
                                //Skip empty lines
                                if (regex_match(toParse,
reg_rules[int(instGenericType::noOps)]))
                                {
                                        continue;
                                }
                                instsToFill[counterOnEarth] =
(parseInstruction(toParse, reg_rules, counterOnEarth));
                                counterOnEarth++;
                        }
                        resultsField->AppendText("File has been parsed
correctly\n");
                        cout << "File has been parsed correctly" << endl;
                }
                else
                {
                        cerr << "Invalid use of " << argv[1] << ": Valid
options are \n -replace replaceRuleFile sourceFile outputFile \n -verify regexFile
fileToVerify";
                        throw invalid_argument("Invalid internal use of the
parsing utility");
                }

        }
        catch (exception& ex)
        {
                cerr << "A fatal exception has occured .... Now exiting: " <<
ex.what() << endl;
```

```
                        throw logic_error("A fatal exception has occured using the
parsing utility");
                }
        }
        else if (argc == 5)
        {
                if (string(argv[1]) == string("-replace"))
                {
                        try {
                                vector<string> linesToReplace;
                                vector<pair<string, string>> replaceRules;
                                readFromFileIntoVector(linesToReplace, argv[3], '\n');
                                readFromFileIntoPairVector(replaceRules, argv[2],
'\n');
                                ofstream outs(string(argv[4]).c_str());
                                for (auto& line : linesToReplace)
                                {
                                        for (auto rule : replaceRules)
                                        {
                                                size_t ii;
                                                while ((ii = line.find(rule.first)) !=
string::npos)
                                                {
                                                        line.replace(ii,
rule.first.length(), rule.second);
                                                }
                                        }
                                        outs << line << "\n";
                                }
                                outs.close();
                        }
                        catch (exception& ex)
                        {
                                cerr << "A fatal exception has occured .... Now
exiting: " << ex.what() << endl;
                                throw logic_error("A fatal exception has occured using
the replace utility");
                        }
                }
                else
                {
                        cerr << "Invalid use of " << argv[0] << ": Valid options are
\n -replace replaceRuleFile sourceFile outputFile \n -verify regexFile
fileToVerify";
                        throw invalid_argument("Invalid internal use of the parsing
utility");
                }
        }
        else
        {
                cerr << "Invalid use of " << argv[0] << ": Valid options are \n -
replace replaceRuleFile sourceFile outputFile \n -verify regexFile fileToVerify";
                throw invalid_argument("Invalid internal use of the parsing
utility");
        }
        return EXIT_SUCCESS;
}
```

Datapath and Hazard Detection Related

1) datapath.h

```
#pragma once

void fetch();
void decode();
void execute();
void memory();
void writeBack();
```

2) datapath.cpp

```
#include "instructions.h"
#include <algorithm>
using namespace std;


void fetch()
{
	sort(begin(hazards), end(hazards));
	if (!(binary_search(begin(hazards), end(hazards), 51) ||
binary_search(begin(hazards), end(hazards), 52)
		|| binary_search(begin(hazards), end(hazards), 41) ||
binary_search(begin(hazards), end(hazards), 42)))
	{
		int oldPC = PC;
		PC = updatePC();
		pipeline[3] = pipeline[2];
		pipeline[2] = pipeline[1];
		pipeline[1] = pipeline[0];
		pipeline[0] = inst_memory[oldPC];
	}
	if ((binary_search(begin(hazards), end(hazards), 51)))
	{
		pipeline[3] = pipeline[2];
		pipeline[2] = pipeline[1];
		pipeline[1] = new inst();
	}
	if ((binary_search(begin(hazards), end(hazards), 52)))
	{
		pipeline[3] = pipeline[2];
		pipeline[2] = new inst();
	}
	if ((binary_search(begin(hazards), end(hazards), 41)))
	{
		PC = updatePC();
		pipeline[3] = pipeline[2];
		pipeline[2] = pipeline[1];
		pipeline[1] = pipeline[0];
		pipeline[0] = new inst();
	}

	if ((binary_search(begin(hazards), end(hazards), 42)))
	{
```

```
                PC = updatePC();
                pipeline[3] = pipeline[2];
                pipeline[2] = pipeline[1];
                pipeline[1] = new inst();
                pipeline[0] = new inst();
        }
}

void decode()
{
        pipeline[0]->fetch();
        hazardDetection();
        sort(begin(hazards), end(hazards));
        for_each(begin(hazards), end(hazards), dealWithForwarding);
}

void execute()
{
        pipeline[1]->execute();
}

void memory()
{
        pipeline[2]->memory();
}

void writeBack()
{
        pipeline[3]->writeBack();
}
```

3) hazards.h

```
#pragma once
#include <string>

extern int z;
void hazardDetection();
void dealWithForwarding(int value);

struct hazardMsg
        {
        int value;
        std::string hazard;
        hazardMsg(int i, std::string j) : value(i), hazard(j) {}
        hazardMsg() : hazardMsg(0, std::string("No Hazard")) {}
        };

extern hazardMsg hazardMsgs[14];
```

4) hazards.cpp

```
#include "instructions.h"
#include "hazards.h"
#include <cstring>
#include <algorithm>
```

```
#include <iterator>
#include <string>
using namespace std;

int z;

void hazardDetection()
{
	z = 0;
	int IF_ID_RegRS = -1;
	int IF_ID_RegRT = -1;
	int ID_EX_MemRead = -1;
	int EX_MEM_MEMWr = -1;
	int ID_EX_RegRT = -1;
	int ID_EX_RegRS = -1;
	int ID_EX_RegRD = -1;
	int EX_MEM_RegWrite = -1;
	int EX_MEM_RegRD = -1;
	int MEM_WB_RegWrite = -1;
	int MEM_WB_RegRD = -1;
	int JAL_EXIST = -1;
	int J_EXIST = -1;
	int JR_EXIST = -1;
	int JAL_Where = -1;
	int JALWhere = -1;
	int ID_EX_RegRD_LW = -1;
	int EX_MEM_RegRD_LW = -1;
	int EX_MEM_RegRD_SW = -1;
	int Jal_EXIST = -1;
	int Ret_EXIST = -1;
	int IF_ID_RegRT_JR = -1;
	int IF_ID_RegRT_JAL = -1;
	int JR_Notready = -1;

	// The following loop goes through every pipeline stage and captures what
type of instruction is in the current stage buffer
	memset(hazards, 0, sizeof(hazards));
	for (int i = 0; i < 4; i++)
	{

		rformat* rptr = dynamic_cast<rformat*> (pipeline[i]);
		if (rptr != nullptr)
		{
			if (i == 0) //IF_ID
			{
				EX_MEM_RegWrite = 1;
				MEM_WB_RegWrite = 0;
				IF_ID_RegRS = rptr->rs;
				IF_ID_RegRT = rptr->rt;

			}
			if (i == 1) //ID_EX
			{
				EX_MEM_RegWrite = 1;
				MEM_WB_RegWrite = 0;
				ID_EX_RegRS = rptr->rs;
				ID_EX_RegRT = rptr->rt;
				ID_EX_RegRD = rptr->rd;
```

```
        }

        if (i == 2) // EX_MEM
        {
                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 0;
                EX_MEM_RegRD = rptr->rd;

        }
        if (i == 3) //MEM_WB
        {

                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 1;
                MEM_WB_RegRD = rptr->rd;
        }




}

iformat* iptr = dynamic_cast<iformat*> (pipeline[i]);
if (iptr != nullptr)
{
        if (i == 0)
        {
                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 0;
                IF_ID_RegRS = iptr->rs;
                IF_ID_RegRT = -1;

        }
        if (i == 1)
        {
                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 0;
                ID_EX_RegRS = iptr->rs;
                ID_EX_RegRT = -1;
                ID_EX_RegRD = iptr->rt;
        }

        if (i == 2)
        {
                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 0;
                EX_MEM_RegRD = iptr->rt;

        }
        if (i == 3)
        {

                EX_MEM_RegWrite = 1;
                MEM_WB_RegWrite = 1;
                MEM_WB_RegRD = iptr->rt;
        }

        Lw* Lwptr = dynamic_cast<Lw*> (pipeline[i]);
```

```
if (Lwptr != nullptr)
{
	if (i == 0)
	{
		EX_MEM_RegWrite = 0;
		MEM_WB_RegWrite = 0;
		IF_ID_RegRS = -1;
		IF_ID_RegRT = -1;

		ID_EX_MemRead = 1;

	}
	if (i == 1)
	{
		EX_MEM_RegWrite = 0;
		MEM_WB_RegWrite = 0;
		ID_EX_RegRS = -1;
		ID_EX_RegRT = -1;
		ID_EX_RegRD = -1;

		ID_EX_MemRead = 1;
		ID_EX_RegRD_LW = Lwptr->rt;
	}

	if (i == 2)
	{
		EX_MEM_RegWrite = 0;
		MEM_WB_RegWrite = 0;
		EX_MEM_RegRD = -1;

		ID_EX_MemRead = 1;
		ID_EX_RegRD_LW = -1;
		EX_MEM_RegRD_LW = Lwptr->rt;

	}
	if (i == 3)
	{

		EX_MEM_RegWrite = 0;
		MEM_WB_RegWrite = 1;
		EX_MEM_RegRD_LW = -1;
		MEM_WB_RegRD = Lwptr->rt;
	}
}
Sw* Swptr = dynamic_cast<Sw*> (pipeline[i]);
if (Swptr != nullptr)
{
	if (i == 0)
	{
		EX_MEM_MEMWr = 1;

	}
	if (i == 1)
	{
		EX_MEM_MEMWr = 1;
	}

	if (i == 2)
```

```
                {
                        EX_MEM_MEMWr = 1;
                        EX_MEM_RegRD_SW = Swptr->rt;

                }
                if (i == 3)
                {
                        EX_MEM_MEMWr = 1;
                        EX_MEM_RegRD_SW = -1;
                        MEM_WB_RegRD = Swptr->rt;
                }
        }

        Ble* bleptr = dynamic_cast<Ble*> (pipeline[i]);
        if (bleptr != nullptr)
        {
                if (i == 0) // IF_ID
                {


                        EX_MEM_RegWrite = 0;
                        MEM_WB_RegWrite = 0;
                        IF_ID_RegRS = bleptr->rs;
                        IF_ID_RegRT = bleptr->rt;


                }
                if (i == 1)  // ID_EX
                {
                        EX_MEM_RegWrite = 0;
                        MEM_WB_RegWrite = 0;
                        ID_EX_RegRS = bleptr->rs;
                        ID_EX_RegRT = bleptr->rt;
                }

                if (i == 2) //EX_MEM
                {

                        EX_MEM_RegWrite = 0;
                        MEM_WB_RegWrite = 0;
                        EX_MEM_RegRD = -1;

                }
                if (i == 3)
                {

                        EX_MEM_RegWrite = 0;
                        MEM_WB_RegWrite = 0;
                        MEM_WB_RegRD = -1;
                }

        }
}



/*      Ble* bleptr = dynamic_cast<Ble*> (pipeline[i]);
```

```
if (bleptr != nullptr)
{
if (i == 0) // IF_ID
{
EX_MEM_RegWrite = 0;
MEM_WB_RegWrite = 0;
IF_ID_RegRS = bleptr->rs;
IF_ID_RegRT = bleptr->rt;
}
if (i == 1)  // ID_EX
{
EX_MEM_RegWrite = 0;
MEM_WB_RegWrite = 0;
ID_EX_RegRS = bleptr->rs;
ID_EX_RegRT = bleptr->rt;
ID_EX_RegRD = -1;
}
if (i == 2) //EX_MEM
{
EX_MEM_RegWrite = 0;
MEM_WB_RegWrite = 0;
EX_MEM_RegRD = -1;
}
if (i == 3)
{
EX_MEM_RegWrite = 0;
MEM_WB_RegWrite = 0;
MEM_WB_RegRD = -1;
}
}
*/
J* jptr = dynamic_cast<J*> (pipeline[i]);
if (jptr != nullptr)
{

	Jal* jalptr = dynamic_cast<Jal*> (pipeline[i]);
	if (jalptr != nullptr)
	{
		JAL_EXIST = 1;
		if (i == 0)
		{
			IF_ID_RegRT_JAL = 1;
		}
		if (i == 1)
		{
			EX_MEM_RegWrite = 1;
			MEM_WB_RegWrite = 0;
			JAL_Where = 1;
		}
		if (i == 2)
		{
			EX_MEM_RegWrite = 1;
			MEM_WB_RegWrite = 0;
			JAL_Where = 2;
		}
		if (i == 3)
		{
			EX_MEM_RegWrite = 1;
```

```
                                MEM_WB_RegWrite = 1;
                                JAL_Where = 3;
                        }
                        if (JR_EXIST)
                                JALWhere = JAL_Where;
                }

                if (i == 0)
                {
                        J_EXIST = 1;
                }



        }

        Jr* jrptr = dynamic_cast<Jr*> (pipeline[i]);
        if (jrptr != nullptr)
        {
                if (i == 0)
                {
                        JR_EXIST = 1;
                        IF_ID_RegRT_JR = jrptr->rs;


                }


        }

        Ret* Retptr = dynamic_cast<Ret*> (pipeline[i]);
        if (Retptr != nullptr)
        {
                if (i == 0)
                        Ret_EXIST = 1;
        }
}

//EX and MEM Hazard Read After Write

if (JR_EXIST == 1)
{
        if (IF_ID_RegRT_JR == EX_MEM_RegRD && IF_ID_RegRT_JR != -1)  //
Action = EX_MEM -> IF_ID   Example: add $1,$2,$3 or $2,$4,$5 jr $1
        {
                hazards[z] = 211;
                z++;

        }

        if (IF_ID_RegRT_JR == MEM_WB_RegRD && IF_ID_RegRT_JR != -1) //
Action = MEM_WB -> IF_ID     Example: add $1,$2,$3 or $2,$4,$5 and $5,$9,$10 jr $1
        {
                hazards[z] = 311;
                z++;

        }
```

```
            if (IF_ID_RegRT_JR == ID_EX_RegRD && IF_ID_RegRT_JR != -1) //Stall
nop-> EX   Example: add $1,$2,$3 jr $1
            {
                  hazards[z] = 51;
                  z++;
                  JR_Notready = 1;
            }

            if (IF_ID_RegRT_JR == ID_EX_RegRD_LW && IF_ID_RegRT_JR != -1)
//Stall nop -> Ex  Example: lw $1,20($2) jr $1
            {
                  hazards[z] = 51;
                  z++;
                  JR_Notready = 1;
            }

            if (IF_ID_RegRT_JR == EX_MEM_RegRD_LW && IF_ID_RegRT_JR != -1) //
Stall nop -> M  Example: lw $1,20($2) jr $1
            {
                  hazards[z] = 52;
                  z++;
                  JR_Notready = 1;
            }

            if (JAL_EXIST == 1 && JR_EXIST == 1) // Occurence of JAL and JR
(Multiple usage of $31)
            {
                  if (IF_ID_RegRT_JR == 31)
                  {

                        if (JALWhere == 1) // ID_EX -> IF_ID
                        {
                              hazards[z] = 111;
                              z++;

                        }

                        if (JALWhere == 2) // EX_MEM -> IF_ID
                        {
                              hazards[z] = 211;
                              z++;

                        }

                        if (JALWhere == 3) // MEM_WB -> IF_ID
                        {
                              hazards[z] = 311;
                              z++;

                        }


                  }
            }

      }
```

```
	if (ID_EX_MemRead == 1)  // for example lw $2,20($1) followed by add
$4,$2,$5
	{
		if (EX_MEM_RegRD_LW == ID_EX_RegRS && EX_MEM_RegRD_LW != -1)
		{
			hazards[z] = 52; // Stall nop -> M
			z++;

		}
		else if (EX_MEM_RegRD_LW == ID_EX_RegRT  && EX_MEM_RegRD_LW != -1)
//Stall nop -> M
		{
			hazards[z] = 52;
			z++;
		}

	}

	if (EX_MEM_RegWrite == 1 && ID_EX_MemRead != 1)
	{
		if (EX_MEM_RegRD == ID_EX_RegRS && EX_MEM_RegRD != -1) //EX_MEM ->
ID_EX
		{
			hazards[z] = 221;
			z++;
		}

		if (EX_MEM_RegRD == ID_EX_RegRT && EX_MEM_RegRD != -1) //EX_MEM ->
ID_EX
		{
			hazards[z] = 222;
			z++;


		}

		else EX_MEM_RegWrite = 0;

	}
	if (MEM_WB_RegRD != EX_MEM_RegRD)
	{
		EX_MEM_RegWrite = 0;
	}
	else if (MEM_WB_RegRD == EX_MEM_RegRD || MEM_WB_RegRD == EX_MEM_RegRD_LW)
	{
		EX_MEM_RegWrite = 1;
	}

	if (MEM_WB_RegWrite == 1 && !(EX_MEM_RegWrite))
	{
		if (MEM_WB_RegRD == ID_EX_RegRS && MEM_WB_RegRD != -1) //MEM_WB ->
ID_EX
		{
			hazards[z] = 321;
			z++;
		}
		if (MEM_WB_RegRD == ID_EX_RegRT  && MEM_WB_RegRD != -1) //MEM_WB ->
ID_EX
```

```
		{
			hazards[z] = 322;
			z++;
		}

		if ((MEM_WB_RegRD == EX_MEM_RegRD_SW) && (EX_MEM_MEMWr == 1) &&
(MEM_WB_RegRD != -1)) // MEM_WB -> EX_MEM   Example: add $1,$3,$2  sw $1,20($2)
		{
			hazards[z] = 332;
			z++;
		}



	}
	/*if (JAL_EXIST) // Occurence of JAL and usage of $31 in other instructions
	{
	if (EX_MEM_RegWrite)
	{
	if (ID_EX_RegRS == 31) // Rs is using $31  //EX_MEM -> ID_EX
	{
	hazards[z] = 221;
	z++;
	}
	if (ID_EX_RegRT == 31) // RT is using $31
	{
	hazards[z] = 222;
	z++;
	}
	}
	if (MEM_WB_RegWrite)
	{
	if (ID_EX_RegRS == 31) // Rs is using $31 //MEM_WB -> ID_EX
	{
	hazards[z] = 321;
	z++;
	}
	if (ID_EX_RegRT == 31) // RT is using $31
	{
	hazards[z] = 322;
	z++;
	}
	}
	}*/


	Ble* bleptrSpecial = dynamic_cast<Ble*> (pipeline[0]);
	if (bleptrSpecial != nullptr)
	{
		if (EX_MEM_RegRD == ID_EX_RegRS && EX_MEM_RegRD != -1)
		{
			hazards[z] = 221;
			z++;
		}
		if (EX_MEM_RegRD == ID_EX_RegRT && EX_MEM_RegRD != -1)
		{
			hazards[z] = 222;
			z++;
```

```
                }
                if (MEM_WB_RegRD == ID_EX_RegRS  && MEM_WB_RegRD != -1)
                {
                        hazards[z] = 321;
                        z++;
                }
                if (MEM_WB_RegRD == ID_EX_RegRT && MEM_WB_RegRD != -1)
                {
                        hazards[z] = 322;
                        z++;
                }

                if (EX_MEM_RegRD == IF_ID_RegRS && EX_MEM_RegRD != -1)
                {
                        hazards[z] = 211;
                        z++;
                }
                if (EX_MEM_RegRD == IF_ID_RegRT && EX_MEM_RegRD != -1)
                {
                        hazards[z] = 212;
                        z++;
                }
                if (MEM_WB_RegRD == IF_ID_RegRS  && MEM_WB_RegRD != -1)
                {
                        hazards[z] = 311;
                        z++;
                }
                if (MEM_WB_RegRD == IF_ID_RegRT && MEM_WB_RegRD != -1)
                {
                        hazards[z] = 312;
                        z++;
                }

        }

        if ((IF_ID_RegRT_JR != -1 && JR_Notready != 1) || IF_ID_RegRT_JAL == 1 ||
Ret_EXIST == 1 || J_EXIST == 1) //Flush F
        {
                int reallyFlush = 0;

                J* jptr = dynamic_cast <J*> (pipeline[0]);
                Jr* jrptr = dynamic_cast <Jr*> (pipeline[0]);
                Ret* retptr = dynamic_cast <Ret*> (pipeline[0]);

                if (jptr)
                        if (jptr->address != PC)
                                reallyFlush = 1;

                if (retptr)
                        if (retptr->addressPopped != PC)
                                reallyFlush = 1;

                if (jrptr)
                        if (jrptr->rsData != PC)
                                reallyFlush = 1;

                if (reallyFlush)
                {
```

```
                        hazards[z] = 41;
                        z++;
                }
        }
}

void dealWithForwarding(int value)
{
        switch (value)
        {
        case 111:
        {
                rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[1]);
                iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[1]);
                Jal* jalptrfrom = dynamic_cast<Jal*> (pipeline[1]);
                Jr* jrptr = dynamic_cast<Jr*> (pipeline[0]);
                if (rptrfrom != nullptr)
                {
                        jrptr->rsData = rptrfrom->writeData;
                }
                else if (iptrfrom != nullptr)
                {
                        jrptr->rsData = iptrfrom->writeData;
                }
                else if (jalptrfrom != nullptr)
                {
                        jrptr->rsData = jalptrfrom->returnAddress;
                }
                break;
        }
        case 211:
        {
                rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[2]);
                iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[2]);
                Jal* jalptrfrom = dynamic_cast<Jal*> (pipeline[2]);
                Jr* jrptr = dynamic_cast<Jr*> (pipeline[0]);
                Ble* bleptr = dynamic_cast<Ble*> (pipeline[0]);
                if (rptrfrom != nullptr)
                {
                        if (jrptr != nullptr)
                                jrptr->rsData = rptrfrom->writeData;
                        if (bleptr != nullptr)
                                bleptr->rsData = rptrfrom->writeData;
                }
                else if (iptrfrom != nullptr)
                {
                        if (jrptr != nullptr)
                                jrptr->rsData = iptrfrom->writeData;
                        if (bleptr != nullptr)
                                bleptr->rsData = iptrfrom->writeData;
                }
                else if (jalptrfrom != nullptr)
                {
                        if (jrptr != nullptr)
                                jrptr->rsData = jalptrfrom->returnAddress;
                        if (bleptr != nullptr)
                                bleptr->rsData = jalptrfrom->returnAddress;
                }
```

```
        break;
}
case 212:
{
        rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[2]);
        iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[2]);
        Jal* jalptrfrom = dynamic_cast<Jal*> (pipeline[2]);
        Ble* bleptr = dynamic_cast<Ble*> (pipeline[0]);
        if (rptrfrom != nullptr)
        {
                if (bleptr != nullptr)
                        bleptr->rtData = rptrfrom->writeData;
        }
        else if (iptrfrom != nullptr)
        {
                if (bleptr != nullptr)
                        bleptr->rtData = iptrfrom->writeData;
        }
        else if (jalptrfrom != nullptr)
        {
                if (bleptr != nullptr)
                        bleptr->rtData = jalptrfrom->returnAddress;
        }
        break;
}
case 311:
{
        rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[3]);
        iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[3]);
        Jal* jalptrfrom = dynamic_cast<Jal*> (pipeline[3]);
        Jr* jrptr = dynamic_cast<Jr*> (pipeline[0]);
        Ble* bleptr = dynamic_cast<Ble*> (pipeline[0]);
        if (rptrfrom != nullptr)
        {
                if (jrptr != nullptr)
                        jrptr->rsData = rptrfrom->writeData;
                if (bleptr != nullptr)
                        bleptr->rsData = rptrfrom->writeData;
        }
        else if (iptrfrom != nullptr)
        {
                if (jrptr != nullptr)
                        jrptr->rsData = iptrfrom->writeData;
                if (bleptr != nullptr)
                        bleptr->rsData = iptrfrom->writeData;
        }
        else if (jalptrfrom != nullptr)
        {
                if (jrptr != nullptr)
                        jrptr->rsData = jalptrfrom->returnAddress;
                if (bleptr != nullptr)
                        bleptr->rsData = jalptrfrom->returnAddress;
        }
        break;
}
case 312:
{
        rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[3]);
```

```
		iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[3]);
		Jal* jalptrfrom = dynamic_cast<Jal*> (pipeline[3]);
		Ble* bleptr = dynamic_cast<Ble*> (pipeline[0]);
		if (rptrfrom != nullptr)
		{
			if (bleptr != nullptr)
				bleptr->rtData = rptrfrom->writeData;
		}
		else if (iptrfrom != nullptr)
		{
			if (bleptr != nullptr)
				bleptr->rtData = iptrfrom->writeData;
		}
		else if (jalptrfrom != nullptr)
		{
			if (bleptr != nullptr)
				bleptr->rtData = jalptrfrom->returnAddress;
		}
		break;
	}
	case 221:
	{
		rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[2]);
		iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[2]);
		rformat* rptrto = dynamic_cast<rformat*> (pipeline[1]);
		iformat* iptrto = dynamic_cast<iformat*> (pipeline[1]);
		if (rptrfrom != nullptr)
		{
			if (rptrto != nullptr)
			{
				rptrto->rsData = rptrfrom->writeData;
			}
			else if (iptrto != nullptr)
			{
				iptrto->rsData = rptrfrom->writeData;
			}
		}
		else if (iptrfrom != nullptr)
		{
			if (rptrto != nullptr)
			{
				rptrto->rsData = iptrfrom->writeData;
			}
			else if (iptrto != nullptr)
			{
				iptrto->rsData = iptrfrom->writeData;
			}
		}
		break;
	}
	case 222:
	{
		rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[2]);
		iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[2]);
		iformat* iptrto = dynamic_cast<iformat*> (pipeline[1]);
		rformat* rptrto = dynamic_cast<rformat*> (pipeline[1]);
		if (rptrfrom != nullptr)
		{
```

```
                if (rptrto != nullptr)
                {
                        rptrto->rtData = rptrfrom->writeData;
                }
                else if (iptrto != nullptr)
                {
                        iptrto->rtData = rptrfrom->writeData;
                }
        }
        else if (iptrfrom != nullptr)
        {
                if (rptrto != nullptr)
                {
                        rptrto->rtData = iptrfrom->writeData;
                }
                else if (iptrto != nullptr)
                {
                        iptrto->rtData = iptrfrom->writeData;
                }
        }
        break;
}
case 321:
{
        rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[3]);
        iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[3]);
        rformat* rptrto = dynamic_cast<rformat*> (pipeline[1]);
        iformat* iptrto = dynamic_cast<iformat*> (pipeline[1]);
        if (rptrfrom != nullptr)
        {
                if (rptrto != nullptr)
                {
                        rptrto->rsData = rptrfrom->writeData;
                }
                else if (iptrto != nullptr)
                {
                        iptrto->rsData = rptrfrom->writeData;
                }
        }
        else if (iptrfrom != nullptr)
        {
                if (rptrto != nullptr)
                {
                        rptrto->rsData = iptrfrom->writeData;
                }
                else if (iptrto != nullptr)
                {
                        iptrto->rsData = iptrfrom->writeData;
                }
        }
        break;
}
case 322:
{
        rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[3]);
        iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[3]);
        rformat* rptrto = dynamic_cast<rformat*> (pipeline[1]);
        iformat* iptrto = dynamic_cast<iformat*> (pipeline[1]);
```

```
                if (rptrfrom != nullptr)
                {
                        if (rptrto != nullptr)
                        {
                                rptrto->rtData = rptrfrom->writeData;
                        }
                        else if (iptrto != nullptr)
                        {
                                iptrto->rtData = rptrfrom->writeData;
                        }
                }
                else if (iptrfrom != nullptr)
                {
                        if (rptrto != nullptr)
                        {
                                rptrto->rtData = iptrfrom->writeData;
                        }
                        else if (iptrto != nullptr)
                        {
                                iptrto->rtData = iptrfrom->writeData;
                        }
                }
                break;
        }
        case 332:
        {
                rformat* rptrfrom = dynamic_cast<rformat*> (pipeline[3]);
                iformat* iptrfrom = dynamic_cast<iformat*> (pipeline[3]);
                iformat* iptrto = dynamic_cast<iformat*> (pipeline[2]);
                if (rptrfrom != nullptr)
                {
                        iptrto->rtData = rptrfrom->writeData;
                }
                else if (iptrfrom != nullptr)
                {
                        iptrto->rtData = iptrfrom->writeData;
                }
                break;
        }
        default:
                break;
        }
}

hazardMsg hazardMsgs[14] =
{
        hazardMsg(41, string("Need to flush fetched instruction during next
cycle.")),
        hazardMsg(42, string("Need to flush fetched and decoded instructions during
next cycle.")),
        hazardMsg(51, string("Need to stall at the decode stage for the next
cycle.")),
        hazardMsg(52, string("Need to stall at the execute stage for the next
cycle.")),
        hazardMsg(111, string("Forwarded the value of rs from execute to
decode.")),
        hazardMsg(211, string("Forwarded the value of rs from memory to decode.")),
        hazardMsg(212, string("Forwarded the value of rt from memory to decode.")),
```

```
        hazardMsg(311, string("Forwarded the value of rs from writeback to
decode.")),
        hazardMsg(312, string("Forwarded the value of rt from writeback to
decode.")),
        hazardMsg(221, string("Forwarded the value of rs from memory to
execute.")),
        hazardMsg(222, string("Forwarded the value of rt from memory to
execute.")),
        hazardMsg(321, string("Forwarded the value of rs from writeback to
execute.")),
        hazardMsg(322, string("Forwarded the value of rt from writeback to
execute.")),
        hazardMsg(332, string("Forwarded the value of rt from writeback to
memory."))
};
```

Instructions Related (only important instructions here, rest can be viewed on the GitHub repo, check project overview)

1) inst.h

```
#pragma once

#include "utility.h"
class inst
{
public:
        inst();
        inst(std::string _instString);
        virtual ~inst();
        virtual void fetch();
        virtual void execute();
        virtual void memory();
        virtual void writeBack();
        virtual bool valid();
        std::string instString;
};
```

2) inst.cpp

```
#include "inst.h"
#include <stdexcept>
using namespace std;

inst::inst():inst("nop")
{}

inst::inst(string _instString = "nop"): instString(_instString){}
inst::~inst() {}
void inst::fetch() {}
void inst::execute() {}
void inst::memory() {}
void inst::writeBack() {}
bool inst::valid() { return true; }
```

3) rformat.h

```
#pragma once
#include "inst.h"
#include <string>

class rformat : public inst
{
public:
	rformat(int _rd = -1, int _rs = -1,  int _rt = -1, int _instAddress = -1,
std::string _instString = "nop");
	virtual ~rformat();
	virtual void fetch();
	virtual void writeBack();
	int rs;
	int rt;
	int rd;
	int rsData;
	int rtData;
	int writeData;
	int instAddress;
	virtual bool valid();
};
```

4) rformat.cpp

```
#include "rformat.h"
#include <stdexcept>
using namespace std;

rformat::rformat(int _rd, int _rs, int _rt,  int _instAddress, string _instString)
	:inst(_instString), instAddress(_instAddress) , rs(_rs), rt(_rt), rd(_rd)
{
	if (!valid()) throw logic_error("Bad construction of rformat
instruction.");
}

rformat::~rformat() {  }

bool rformat::valid()
{
	return (rs >= 0) && (rs <= 31)
		&& (rt >= 0) && (rt <= 31)
		&& (rd >= 0) && (rd <= 31)
		&& (instAddress >= 0);
}


void rformat::fetch()
{
	rsData = readRegister(rs);
	rtData = readRegister(rt);
}

void rformat::writeBack()
{
	writeRegister(rd, writeData);
}
```

5) iformat.h

```
#pragma once
#include "inst.h"
#include <string>

class iformat : public inst
{
public:
	iformat( int _rt = -1, int _rs = -1, int _immediate = 0, int _instAddress =
-1, std::string _instString = "nop");
	virtual ~iformat();
	virtual void fetch();
	virtual void writeBack();
	int rs;
	int rt;
	int immediate;
	int rsData;
	int rtData;
	int writeData;
	int instAddress;
	virtual bool valid();
};
```

6) iformat.cpp

```
#include "iformat.h"
#include <stdexcept>
using namespace std;

iformat::iformat(int _rt, int _rs,  int _immediate, int _instAddress, string
_instString)
	:inst(_instString), instAddress(_instAddress) , rs(_rs), rt(_rt),
immediate(_immediate)
{
	if (!valid()) throw logic_error("Bad construction of iformat
instruction.");
}

iformat::~iformat() {  }

void iformat::fetch()
{
	rsData = readRegister(rs);
	rtData = readRegister(rt);
}

void iformat::writeBack()
{
	writeRegister(rt, writeData);
}

bool iformat::valid()
{
	return (rs >= 0) && (rs<=31)
		&& (rt >= 0) && (rt <= 31)
		&& (instAddress >= 0);
}
```

5) Ble.h

```
#pragma once
#include "iformat.h"

class Ble : public iformat
{
public:
	Ble(int _rt = -1, int _rs = -1, int _immediate = 0, int _instAddress = -1,
std::string _instString = "nop");
	virtual ~Ble();
	virtual void fetch();
	virtual void execute();
	virtual void writeBack();

	int addressTrue;
	int addressIfNotTaken;
	int addressIfTaken;
	int wrongPredicitonFlag;
};
```

6) Ble.cpp

```
#include "Ble.h"
#include <algorithm>
using namespace std;


Ble::Ble(int _rt, int _rs, int _immediate, int _instAddress, string _instString)
	:iformat(_rt, _rs, _immediate, _instAddress, _instString) {}

Ble::~Ble() {}

void Ble::fetch()
{
	rsData = readRegister(rs);
	rtData = readRegister(rt);
	addressIfNotTaken = PC;
	addressIfTaken = immediate + PC;
}

void Ble::execute()
{
	if (rsData <= rtData)
	{
		addressTrue = addressIfTaken;
	}
	else
	{
		addressTrue = addressIfNotTaken;
	}

	if (right_prediction() == false)
	{
		hazards[z] = 42;
		z++;
```

```
		int* it = find(begin(hazards), end(hazards), 41);
		if (it != end(hazards))
		{
			*it = 0;
		}
		it = find(begin(hazards), end(hazards), 51);
		if (it != end(hazards))
		{
			*it = 0;
		}

		it = find(begin(hazards), end(hazards), 52);
		if (it != end(hazards))
		{
			*it = 0;
		}
		sort(begin(hazards), end(hazards));
	}
}

void Ble::writeBack() {}
```

7) Jr.h

```
#pragma once
#include "inst.h"
#include <string>

class Jr : public inst
{
public:
	Jr(int _rs = -1, int _instAddress = -1, std::string _instString = "nop");
	virtual ~Jr();
	virtual void fetch();

	int rs;
	int rsData;
	int instAddress;

	virtual bool valid();
};
```

8) Jr.cpp

```
#pragma once
#include "inst.h"
#include <string>

class Jr : public inst
{
public:
	Jr(int _rs = -1, int _instAddress = -1, std::string _instString = "nop");
	virtual ~Jr();
	virtual void fetch();

	int rs;
	int rsData;
	int instAddress;
```

```
	virtual bool valid();
};
```

9) J.h

```
#pragma once
#include "inst.h"
#include <string>

class J : public inst
{
public:
	J(int _immediate = -1, int _instAddress = -1, std::string _instString =
"nop");
	virtual ~J();

	int immediate;
	int address;
	int instAddress;

	virtual void fetch();
	virtual bool valid();
};
```

10) J.cpp

```
#include "J.h"
#include <stdexcept>
using namespace std;



J::J(int _immediate, int _instAddress, string _instString)
	: inst(_instString), instAddress(_instAddress), immediate(_immediate)
{
	if (!valid()) throw logic_error("Bad construction of a jump instruction.");
}

J::~J() {}

void J::fetch()
{
	int tempPC = PC >> 26;
	tempPC = tempPC << 26;
	address = immediate  | tempPC;
}

bool J::valid()
{
	return (immediate >=0)
		&& (instAddress >= 0);
}
```

11) Ret.h

```
#pragma once
#include "inst.h"
#include <string>

class Ret : public inst
{
public:
	Ret(int _instAddress = -1, std::string _instString = "nop");
	virtual ~Ret();
	void fetch();
	int addressPopped;
	int instAddress;
	virtual bool valid();
};
```

12) Ret.cpp

```
#include "Ret.h"
#include <stdexcept>
using namespace std;


Ret::Ret(int _instAddress, string _instString): inst(_instString),
instAddress(_instAddress)
{
	if (!valid()) throw logic_error("Bad construction of ret instruction.");
}

Ret::~Ret() {}

void Ret::fetch() { addressPopped = popfromstack(); }

bool Ret::valid()
{
	return (instAddress >= 0);
}
```