University of California, Riverside

Department of Computer Science and Engineering

Title: Lab 1 Report

Course: Advanced Computer Architecture

Student Name: Mahbod Afarin

Student Number: 862186340

January 2020

1- Implementing forwarding

First, I implemented forwarding in the Pipesim. In the below piece of code, I showed the changes in Pipesim for implementing forwarding. I added forwarding in the pipeline.cpp. The whole code is attached to this report and also it is on the appendix of this report.

```
bool Pipeline::hasDependency(void)
{
    if (forwarding) //adding forwading
    {
        if (pipeline[DECODE].inst->type == NOP)
            return false;

        for (int i = EXEC; i < WB; i++)
        {

            if (pipeline[i].inst == NULL)
                continue;

            if (pipeline[i].inst->type == NOP)
                continue;

            if (pipeline[EXEC].inst->type == LW)
            {
                if ((pipeline[i].inst->dest != -1) &&
                    (pipeline[i].inst->dest == pipeline[DECODE].inst->src1 ||
                     pipeline[i].inst->dest == pipeline[DECODE].inst->src2))
                {
                    return true;
                }
            }
        }

        return false;
    }
    else
    {

        if (pipeline[DECODE].inst->type == NOP)
            return false;
```

Figure 1: Implementing forwarding.

In the following pictures I showed the results of the execution of instruction.txt, instruction2.txt, and instruction3.txt without forwarding. The clock cycles for the instruction.txt, instruction2.txt, and instruction3.txt is 18, 23, and 16 respectively.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction.txt
Loading application...instruction.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SUB r2 r3 r4
MULT r3 r1 r5
DIV r4 r3 r6
LW r5 r4
SW r5 r7
BNEZ r7 r8
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SUB r2 r3 r4    ADD r1 r2 r3    *               *               *
3       MULT r3 r1 r5   SUB r2 r3 r4    ADD r1 r2 r3    *               *
4       DIV r4 r3 r6    MULT r3 r1 r5   SUB r2 r3 r4    ADD r1 r2 r3    *
5       DIV r4 r3 r6    MULT r3 r1 r5   *               SUB r2 r3 r4    ADD r1 r2 r3
6       LW r5 r4        DIV r4 r3 r6    MULT r3 r1 r5   *               SUB r2 r3 r4
7       LW r5 r4        DIV r4 r3 r6    *               MULT r3 r1 r5   *
8       LW r5 r4        DIV r4 r3 r6    *               *               MULT r3 r1 r5
9       SW r5 r7        LW r5 r4        DIV r4 r3 r6    *               *
10      SW r5 r7        LW r5 r4        *               DIV r4 r3 r6    *
11      SW r5 r7        LW r5 r4        *               *               DIV r4 r3 r6
12      BNEZ r7 r8      SW r5 r7        LW r5 r4        *               *
13      BNEZ r7 r8      SW r5 r7        *               LW r5 r4        *
14      BNEZ r7 r8      SW r5 r7        *               *               LW r5 r4
15      *               BNEZ r7 r8      SW r5 r7        *               *
16      *               *               BNEZ r7 r8      SW r5 r7        *
17      *               *               *               BNEZ r7 r8      SW r5 r7
18      *               *               *               *               BNEZ r7 r8
19      *               *               *               *               *
Completed in 18 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 2: The result of the instruction.txt without forwarding

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction2.txt
Loading application...instruction2.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SW r1 r2
LW r7 r2
ADD r5 r7 r1
LW r8 r2
SW r7 r8
ADD r8 r8 r2
LW r9 r8
SW r9 r8
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SW r1 r2        ADD r1 r2 r3    *               *               *
3       LW r7 r2        SW r1 r2        ADD r1 r2 r3    *               *
4       LW r7 r2        SW r1 r2        *               ADD r1 r2 r3    *
5       LW r7 r2        SW r1 r2        *               *               ADD r1 r2 r3
6       ADD r5 r7 r1    LW r7 r2        SW r1 r2        *               *
7       LW r8 r2        ADD r5 r7 r1    LW r7 r2        SW r1 r2        *
8       LW r8 r2        ADD r5 r7 r1    *               LW r7 r2        SW r1 r2
9       LW r8 r2        ADD r5 r7 r1    *               *               LW r7 r2
10      SW r7 r8        LW r8 r2        ADD r5 r7 r1    *               *
11      ADD r8 r8 r2    SW r7 r8        LW r8 r2        ADD r5 r7 r1    *
12      ADD r8 r8 r2    SW r7 r8        *               LW r8 r2        ADD r5 r7 r1
13      ADD r8 r8 r2    SW r7 r8        *               *               LW r8 r2
14      LW r9 r8        ADD r8 r8 r2    SW r7 r8        *               *
15      SW r9 r8        LW r9 r8        ADD r8 r8 r2    SW r7 r8        *
16      SW r9 r8        LW r9 r8        *               ADD r8 r8 r2    SW r7 r8
17      SW r9 r8        LW r9 r8        *               *               ADD r8 r8 r2
18      *               SW r9 r8        LW r9 r8        *               *
19      *               SW r9 r8        *               LW r9 r8        *
20      *               SW r9 r8        *               *               LW r9 r8
21      *               *               SW r9 r8        *               *
22      *               *               *               SW r9 r8        *
23      *               *               *               *               SW r9 r8
24      *               *               *               *               *
Completed in 23 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 3: The result of the instruction2.txt without forwarding

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction3.txt
Loading application...instruction3.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SUB r2 r3 r1
ADD r4 r3 r1
SUB r3 r1 r5
LW r6 r3
SW r6 r4
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SUB r2 r3 r1    ADD r1 r2 r3    *               *               *
3       ADD r4 r3 r1    SUB r2 r3 r1    ADD r1 r2 r3    *               *
4       ADD r4 r3 r1    SUB r2 r3 r1    *               ADD r1 r2 r3    *
5       ADD r4 r3 r1    SUB r2 r3 r1    *               *               ADD r1 r2 r3
6       SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1    *               *
7       LW r6 r3        SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1    *
8       SW r6 r4        LW r6 r3        SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1
9       SW r6 r4        LW r6 r3        *               SUB r3 r1 r5    ADD r4 r3 r1
10      SW r6 r4        LW r6 r3        *               *               SUB r3 r1 r5
11      *               SW r6 r4        LW r6 r3        *               *
12      *               SW r6 r4        *               LW r6 r3        *
13      *               SW r6 r4        *               *               LW r6 r3
14      *               *               SW r6 r4        *               *
15      *               *               *               SW r6 r4        *
16      *               *               *               *               SW r6 r4
17      *               *               *               *               *
Completed in 16 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 4: The result of the instruction3.txt without forwarding.

In the following pictures I showed the results of the instruction.txt, instruction2.txt, and instruction3.txt with forwarding. The clock cycles in forwarding mood for the instruction.txt, instruction2.txt, and instruction3.txt is 12, 16, and 11 respectively.

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction.txt -f
Loading application...instruction.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SUB r2 r3 r4
MULT r3 r1 r5
DIV r4 r3 r6
LW r5 r4
SW r5 r7
BNEZ r7 r8
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SUB r2 r3 r4    ADD r1 r2 r3    *               *               *
3       MULT r3 r1 r5   SUB r2 r3 r4    ADD r1 r2 r3    *               *
4       DIV r4 r3 r6    MULT r3 r1 r5   SUB r2 r3 r4    ADD r1 r2 r3    *
5       LW r5 r4        DIV r4 r3 r6    MULT r3 r1 r5   SUB r2 r3 r4    ADD r1 r2 r3
6       SW r5 r7        LW r5 r4        DIV r4 r3 r6    MULT r3 r1 r5   SUB r2 r3 r4
7       BNEZ r7 r8      SW r5 r7        LW r5 r4        DIV r4 r3 r6    MULT r3 r1 r5
8       BNEZ r7 r8      SW r5 r7        *               LW r5 r4        DIV r4 r3 r6
9       *               BNEZ r7 r8      SW r5 r7        *               LW r5 r4
10      *               *               BNEZ r7 r8      SW r5 r7        *
11      *               *               *               BNEZ r7 r8      SW r5 r7
12      *               *               *               *               BNEZ r7 r8
13      *               *               *               *               *
Completed in 12 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 5: The result of the instruction.txt with forwarding.

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction2.txt -f
Loading application...instruction2.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SW r1 r2
LW r7 r2
ADD r5 r7 r1
LW r8 r2
SW r7 r8
ADD r8 r8 r2
LW r9 r8
SW r9 r8
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SW r1 r2        ADD r1 r2 r3    *               *               *
3       LW r7 r2        SW r1 r2        ADD r1 r2 r3    *               *
4       ADD r5 r7 r1    LW r7 r2        SW r1 r2        ADD r1 r2 r3    *
5       LW r8 r2        ADD r5 r7 r1    LW r7 r2        SW r1 r2        ADD r1 r2 r3
6       LW r8 r2        ADD r5 r7 r1    *               LW r7 r2        SW r1 r2
7       SW r7 r8        LW r8 r2        ADD r5 r7 r1    *               LW r7 r2
8       ADD r8 r8 r2    SW r7 r8        LW r8 r2        ADD r5 r7 r1    *
9       ADD r8 r8 r2    SW r7 r8        *               LW r8 r2        ADD r5 r7 r1
10      LW r9 r8        ADD r8 r8 r2    SW r7 r8        *               LW r8 r2
11      SW r9 r8        LW r9 r8        ADD r8 r8 r2    SW r7 r8        *
12      *               SW r9 r8        LW r9 r8        ADD r8 r8 r2    SW r7 r8
13      *               SW r9 r8        *               LW r9 r8        ADD r8 r8 r2
14      *               *               SW r9 r8        *               LW r9 r8
15      *               *               *               SW r9 r8        *
16      *               *               *               *               SW r9 r8
17      *               *               *               *               *
Completed in 16 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 6: The result of the instruction2.txt with forwarding.

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction3.txt -f
Loading application...instruction3.txt
Read file completed!!
Printing Application:
ADD r1 r2 r3
SUB r2 r3 r1
ADD r4 r3 r1
SUB r3 r1 r5
LW r6 r3
SW r6 r4
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       ADD r1 r2 r3    *               *               *               *
2       SUB r2 r3 r1    ADD r1 r2 r3    *               *               *
3       ADD r4 r3 r1    SUB r2 r3 r1    ADD r1 r2 r3    *               *
4       SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1    ADD r1 r2 r3    *
5       LW r6 r3        SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1    ADD r1 r2 r3
6       SW r6 r4        LW r6 r3        SUB r3 r1 r5    ADD r4 r3 r1    SUB r2 r3 r1
7       *               SW r6 r4        LW r6 r3        SUB r3 r1 r5    ADD r4 r3 r1
8       *               SW r6 r4        *               LW r6 r3        SUB r3 r1 r5
9       *               *               SW r6 r4        *               LW r6 r3
10      *               *               *               SW r6 r4        *
11      *               *               *               *               SW r6 r4
12      *               *               *               *               *
Completed in 11 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 7: The result of the instruction3.txt with forwarding.

2- Questions

2-1- Identify and classify all hazards that exist in instruction.txt and instruction2.txt.

First, I am going to identify the hazards of instraction.txt.

ADD r1 r2 r3	R1 ← r2 + r3
SUB r2 r3 r4	R2 ← r3 + r4
MULT r3 r1 r5	R3 ← r1 * r5
DIV r4 r3 r6	R4 ← r3 / r6
LW r5 r4	R5 ← mem [r4]
SW r5 r7	Mem [r7] ← r5
BNEZ r7 r8	If (r7 != r8) go to [address]

1- Data Hazards:

1- Read After Write (RAW) between ADD and MULT instructions in r1 register.
ADD r1 r2 r3
MULT r3 r1 r5

2- Read After Write (RAW) between MULT and DIV instructions in r3 register.
MULT r3 r1 r5
DIV r4 r3 r6

3- Read After Write (RAW) between DIV and LW instructions in r4 register.
DIV r4 r3 r6
LW r5 r4

4- Read After Write (RAW) between LW and SW instructions in r5 register.
LW r5 r4
SW r5 r7

5- Write After Read (WAR) between ADD and SUB instructions in r2 register. **WAR is not a true dependency in pipeline and it is anti-dependency.**
ADD r1 r2 r3
SUB r2 r3 r4

6- Write After Read (WAR) between SUB and MULT instructions in r3 register. **WAR is not a true dependency in pipeline and it is anti-dependency.**
SUB r2 r3 r4
MULT r3 r1 r5

2- Structural Hazards:

In this example our instruction cache and data cache are separate, so we have not structural hazard. Structural hazard happens when we have conflict for use of a resource. In five stage pipeline we need to access to the same cache in instruction fetch stage and memory stage, if we don't separate these two caches.

3- Control Hazard:

Control hazard occurs when we have a branch instruction because fetching next instruction depends on a branch outcome. In this example if we have instructions after the BNEZ r7 r8, we faced control hazard. **So, We have not control hazard because we have not any instructions after BNEZ r7 r8.**

Second, I am going to identify the hazards of instraction2.txt.

ADD r1 r2 r3	r1 ← r2 + r3
SW r1 r2	mem [r2] ← r1
LW r7 r2	r7 ← mem [r2]
ADD r5 r7 r1	r5 ← r7 + r1
LW r8 r2	r8 ← mem [r2]
SW r7 r8	mem [r8] ← r7
ADD r8 r8 r2	r8 ← r8 + r2
LW r9 r8	r9 ← mem [r8]
SW r9 r8	r8 ← mem [r9]

1- Data Hazards:

1- Read After Write (RAW) hazard between ADD and SW instructions in r1 register.
ADD r1 r2 r3
SW r1 r2

2- Read After Write (RAW) hazard between LW and ADD instructions in r7 register.
LW r7 r2
ADD r5 r7 r1

3- Read After Write (RAW) hazard between LW and SW instructions in r8 register.
LW r8 r2
SW r7 r8

4- Read After Write (RAW) hazard between ADD and LW instructions in r8 register.
ADD r8 r8 r2
LW r9 r8

5- Read After Write (RAW) hazard between LW and SW instructions in r9 register.
LW r9 r8
SW r9 r8

6- Read After Write (RAW) hazard between LW and ADD instructions in r8 register.
LW r8 r2
ADD r8 r8 r2

7- Write After Read (WAR) hazard between SW and ADD instructions in r8 register. **WAR is not a true dependency in pipeline and it is anti-dependency.**
SW r7 r8
ADD r8 r8 r2

8- Write After Read (WAR) hazard between LW and SW instructions in r8 register. **WAR is not a true dependency in pipeline and it is anti-dependency.**
LW r9 r8
SW r9 r8

2- Structural Hazards:

In this example our instruction cache and data cache are separate, so we have not structural hazard. Structural hazard happens when we have conflict for use of a resource. In five stage pipeline we need to access to the same cache in instruction fetch stage and memory stage, if we don't separate these two caches.

3- Control Hazard:

We don't have control hazard in this example.

2-2- What is the CPI for instruction.txt and instruction2.txt with no forwarding? What is the CPI with forwarding?
What is the speedup for instruction.txt and instruction2.txt when using full forwarding?

CPI for instruction.txt with no forwarding:

$$CPI = \frac{Clock\ Cycles}{Instructions} = \frac{18}{7} = 2.57142$$

CPI for instruction2.txt with no forwarding:

$$CPI = \frac{Clock\ Cycles}{Instructions} = \frac{23}{9} = 2.55555$$

CPI for instruction.txt with forwarding:

$$CPI = \frac{Clock\ Cycles}{Instructions} = \frac{12}{7} = 1.71428$$

CPI for instruction2.txt with forwarding:

$$CPI = \frac{Clock\ Cycles}{Instructions} = \frac{16}{9} = 1.77777$$

Speedup for instraction.txt:

$$Speedup = \frac{Execution\ Time(no\ forwarding)}{Execution\ Time(forwarding)} = \frac{2.57142}{1.71428} = 1.5$$

Speedup for instraction2.txt:

$$Speedup = \frac{Execution\ Time(no\ forwarding)}{Execution\ Time(forwarding)} = \frac{2.55555}{1.77777} = 1.4375$$

2-3- Using a pipeline without forwarding, reorder the instructions in instruction2.txt to minimize the number of stalls. Show the new reordered instruction trace and calculate the speedup compared to the original instruction trace.

With the below order of instruction, we can have 16 cycles instead of 23 instructions.

LW r8 r2
LW r7 r2
ADD r1 r2 r3
SW r7 r8
ADD r8 r8 r2
SW r1 r2
ADD r5 r7 r1
LW r9 r8
SW r9 r8

$$Speedup = \frac{Execution\ (old)}{Execution\ (new)} = \frac{23}{16} = 1.4375$$

The following picture show the result of the reordering trace execution.

```
mahbod@mahbod:~/Desktop/Pipesim$ ./pipesim -i instruction2.txt
Loading application...instruction2.txt
Read file completed!!
Printing Application:
LW r8 r2
LW r7 r2
ADD r1 r2 r3
SW r7 r8
ADD r8 r8 r2
SW r1 r2
ADD r5 r7 r1
LW r9 r8
SW r9 r8
Initializing pipeline...
Cycle   IF              ID              EXEC            MEM             WB
0       *               *               *               *               *
1       LW r8 r2        *               *               *               *
2       LW r7 r2        LW r8 r2        *               *               *
3       ADD r1 r2 r3    LW r7 r2        LW r8 r2        *               *
4       SW r7 r8        ADD r1 r2 r3    LW r7 r2        LW r8 r2        *
5       ADD r8 r8 r2    SW r7 r8        ADD r1 r2 r3    LW r7 r2        LW r8 r2
6       ADD r8 r8 r2    SW r7 r8        *               ADD r1 r2 r3    LW r7 r2
7       SW r1 r2        ADD r8 r8 r2    SW r7 r8        *               ADD r1 r2 r3
8       ADD r5 r7 r1    SW r1 r2        ADD r8 r8 r2    SW r7 r8        *
9       LW r9 r8        ADD r5 r7 r1    SW r1 r2        ADD r8 r8 r2    SW r7 r8
10      SW r9 r8        LW r9 r8        ADD r5 r7 r1    SW r1 r2        ADD r8 r8 r2
11      *               SW r9 r8        LW r9 r8        ADD r5 r7 r1    SW r1 r2
12      *               SW r9 r8        *               LW r9 r8        ADD r5 r7 r1
13      *               SW r9 r8        *               *               LW r9 r8
14      *               *               SW r9 r8        *               *
15      *               *               *               SW r9 r8        *
16      *               *               *               *               SW r9 r8
17      *               *               *               *               *
Completed in 16 cycles
mahbod@mahbod:~/Desktop/Pipesim$
```

Figure 8: The result of the reordering trace

2-4- Would CPI improve from doubling the processor frequency? Why or why not? How can you further improve the CPI of a processor?

$$CPI = \frac{Clock\ Cycles}{Instructions}$$

Base on the above formula, doubling the processor frequency will not improve the CPI because doubling the performance frequency just reduces the cycle time and has not any effects on the number of the cycles of a program.

For improving CPI, we should reduce clock cycles. With minimizing the number of stalls, we can reduce clock cycles. With techniques like forwarding or out of order execution, we can reduce the number of the stalls and as a result, we can improve the CPI. Another way to improve CPI is using wide pipeline or multiple pipeline.

2-5- Assuming a program has 30% of instructions that uses the ALU, and we developed a technique to speedup ALU instructions, what is the maximum speedup that we can achieve?

$$Speedup = \frac{T}{T_e} = \frac{T}{T[(1-f) + \frac{f}{s}]} = \frac{1}{[(1-f) + \frac{f}{s}]} = \frac{1}{[(1-0.3) + \frac{0.3}{s}]}$$

$$We\ have\ maximum\ speedup\ when\ s \to \infty\ , so\ \frac{0.3}{s} \to 0$$

$$\boldsymbol{Speedup = \frac{1}{1 - 0.3} = 1.42857}$$

2-6- If 75% of a program can be done in parallel, given the same execution time what is the speedup of a machine with 4 cores vs a machine with 1 core?

$$\boldsymbol{Speedup = \frac{s + pC}{s + p} = 1 - f + fC = 1 + f(C - 1) = 1 + 0.75 \times (4 - 1) = 3.25}$$

3- Appendix

The pipesim with forwarding feature are available here.

Main.cpp

```
#include <iostream>
#include "unistd.h"
#include "pipeline.h"
#include <stdlib.h>
#include <stdio.h>

using namespace std;

int main(int argc, char *argv[])
{

	int opt;
	bool forwarding = false;
	string fileName = "instruction.txt";
	while ((opt = getopt(argc, argv, "fi:")) != EOF)
		switch (opt)
		{
		case 'f':
			forwarding = true;
			break;
		case 'i':
			fileName.assign(optarg);
			break;
		case '?':
			fprintf(stderr, "usuage is \n -i fileName : to run input file fileName \n -f : for enabling
forwarding ");
		default:
			cout << endl;
			abort();
		}
```

```
	cout << "Loading application..." << fileName << endl;
	Application application;
	application.loadApplication(fileName);
	cout << "Initializing pipeline..." << endl;
	Pipeline pipeline(&application);
	pipeline.forwarding = forwarding;

	do
	{
		pipeline.cycle();
		pipeline.printPipeline();

	} while (!pipeline.done());

	cout << "Completed in " << pipeline.cycleTime - 1 << " cycles" << endl;
	return 0;
}
```

Pipeline.cpp

```
#include "pipeline.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdlib.h>

Register registerFile[16];

Register::Register(void)
{

	dataValue = 0;
	registerNumber = -1;
	registerName = "";
}

Instruction::Instruction(void)
{

	type = NOP;
	dest = -1;
	src1 = -1;
	src2 = -1;
	stage = NONE;
}

Instruction::Instruction(std::string newInst)
{

	std::string buf;
	std::stringstream ss(newInst);
	std::vector<std::string> tokens;

	while (ss >> buf)
```

```
		{
			tokens.push_back(buf);
		}

		if (tokens[0] == "ADD")
			type = ADD;
		else if (tokens[0] == "SUB")
			type = SUB;
		else if (tokens[0] == "MULT")
			type = MULT;
		else if (tokens[0] == "DIV")
			type = DIV;
		else if (tokens[0] == "LW")
			type = LW;
		else if (tokens[0] == "SW")
			type = SW;
		else if (tokens[0] == "BNEZ")
			type = BNEZ;
		else
			type = NOP;

		dest = -1;
		src1 = -1;
		src2 = -1;

		if (tokens.size() > 1)
		{
			dest = atoi(tokens[1].erase(0, 1).c_str());
		}
		if (tokens.size() > 2)
		{
			src1 = atoi(tokens[2].erase(0, 1).c_str());
		}
		if (tokens.size() > 3)
		{
			src2 = atoi(tokens[3].erase(0, 1).c_str());
		}

		// Store and BNEZ has 2 source operands and no destination operand
		if (type == SW || type == BNEZ)
		{
			src2 = src1;
			src1 = dest;
			dest = -1;
		}

		stage = NONE;
}

Application::Application(void)
{

		PC = 0;
}

void Application::loadApplication(std::string fileName)
{

		std::string sLine = "";
```

```
	Instruction *newInstruction;
	std::ifstream infile;
	infile.open(fileName.c_str(), std::ifstream::in);

	if (!infile)
	{
		std::cout << "Failed to open file " << fileName << std::endl;
		return;
	}

	while (!infile.eof())
	{
		getline(infile, sLine);
		if (sLine.empty())
			break;
		newInstruction = new Instruction(sLine);
		instructions.push_back(newInstruction);
	}

	infile.close();
	std::cout << "Read file completed!!" << std::endl;

	printApplication();
}

void Application::printApplication(void)
{

	std::cout << "Printing Application: " << std::endl;
	std::vector<Instruction *>::iterator it;
	for (it = instructions.begin(); it < instructions.end(); it++)
	{

		(*it)->printInstruction();
		std::cout << std::endl;
	}
}

Instruction *Application::getNextInstruction()
{

	Instruction *nextInst = NULL;

	if (PC < instructions.size())
	{
		nextInst = instructions[PC];
		PC += 1;
	}

	if (nextInst == NULL)
		nextInst = new Instruction();

	return nextInst;
}

PipelineStage::PipelineStage(void)
{
	inst = new Instruction();
	stageType = NONE;
```

```
}

void PipelineStage::clear()
{

	inst = NULL;
}

void PipelineStage::process()
{

	// Functionally simulate pipeline stage
	// Since this simulator only models timing, this function currently does nothing
	switch (stageType)
	{
	case FETCH: // Fetch instruction. PC+4
		break;
	case DECODE: // Fetch register operands
		break;
	case EXEC: // Perform ALU operations
		break;
	case MEM: // Load/Store from/to memory
		break;
	case WB: // Writeback result operand to register
		break;
	default:
		break;
	}
}

void PipelineStage::addInstruction(Instruction *newInst)
{

	inst = newInst;
	inst->stage = stageType;
}

Pipeline::Pipeline(Application *app)
{

	pipeline[FETCH].stageType = FETCH;
	pipeline[DECODE].stageType = DECODE;
	pipeline[EXEC].stageType = EXEC;
	pipeline[MEM].stageType = MEM;
	pipeline[WB].stageType = WB;
	cycleTime = 0;

	printPipeline();

	application = app;

	forwarding = false;
}

bool Pipeline::hasDependency(void)
{
	if (forwarding) //adding forwading
	{
		if (pipeline[DECODE].inst->type == NOP)
```

```
                return false;

        for (int i = EXEC; i < WB; i++)
        {

                if (pipeline[i].inst == NULL)
                        continue;

                if (pipeline[i].inst->type == NOP)
                        continue;

                if (pipeline[EXEC].inst->type == LW)
                {
                        if ((pipeline[i].inst->dest != -1) &&
                                (pipeline[i].inst->dest == pipeline[DECODE].inst->src1 ||
                                 pipeline[i].inst->dest == pipeline[DECODE].inst->src2))
                        {
                                return true;
                        }
                }
        }

        return false;
    }
    else
    {

        if (pipeline[DECODE].inst->type == NOP)
                return false;

        // Checks if dependency exist between Decode stage and Exec, Mem stage
        // We assume the register file can read/write in the same cycle so no data dependency exist with
RAW dependency if an instruction is in Decode and WB.
        for (int i = EXEC; i < WB; i++)
        {

                if (pipeline[i].inst == NULL)
                        continue;

                if (pipeline[i].inst->type == NOP)
                        continue;

                if ((pipeline[i].inst->dest != -1) &&
                        (pipeline[i].inst->dest == pipeline[DECODE].inst->src1 ||
                         pipeline[i].inst->dest == pipeline[DECODE].inst->src2))
                {
                        return true;
                }
        }

        return false;
    }
}

void Pipeline::cycle(void)
{

    cycleTime += 1;
    // Check for data hazards
```

```
	// NOTE: Technically, data hazards are detected in the Decode stage. If a data hazard is detected, at the end of the cycle we write 0's (NOP) to the pipeline register so that a NOP will be generated in the EXEC stage in the next cycle.
	// Doing the check here does a dependency check on the instructions in the previous cycle (we haven't advanced the instructions in the pipeline yet). If a dependency exist in the previous cycle, we stall the pipeline in this cycle.
	bool dependencyDetected = hasDependency();

	// WRITEBACK STAGE
	// Mem -> WB Pipeline register
	pipeline[WB].addInstruction(pipeline[MEM].inst);

	// Writeback
	pipeline[WB].process();

	// MEM STAGE
	// Exec -> Mem Pipeline register
	pipeline[MEM].addInstruction(pipeline[EXEC].inst);

	// Mem
	pipeline[MEM].process();

	// EXEC STAGE
	// Decode -> Exec Pipeline register
	// If dependency detected, stall by inserting NOP instruction
	if (!dependencyDetected)
		pipeline[EXEC].addInstruction(pipeline[DECODE].inst);
	else
		pipeline[EXEC].addInstruction(new Instruction());

	// Exec
	pipeline[EXEC].process();

	// DECODE STAGE
	// Fetch -> Decode Pipeline register
	if (!dependencyDetected)
		pipeline[DECODE].addInstruction(pipeline[FETCH].inst);

	// Decode
	pipeline[DECODE].process();

	// FETCH STAGE
	// Fetch
	if (!dependencyDetected)
	{
		pipeline[FETCH].addInstruction(application->getNextInstruction());
		pipeline[FETCH].process();
	}
}

bool Pipeline::done()
{

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

		if (pipeline[i].inst->type != NOP)
			return false;
	}
```

```
	return true;
}

void Pipeline::printPipeline(void)
{

	if (cycleTime == 0)
		std::cout << "Cycle"
					<< "\tIF"
					<< "\t\tID"
					<< "\t\tEXEC"
					<< "\t\tMEM"
					<< "\t\tWB" << std::endl;
	std::cout << cycleTime;
	for (int i = 0; i < 5; i++)
	{

		pipeline[i].printStage();
	}
	std::cout << std::endl;
}

void PipelineStage::printStage(void)
{

	std::cout << "\t";
	inst->printInstruction();
}

void Instruction::printInstruction(void)
{
	if (type == NOP)
		std::cout << instructionNames[type] << "	";
	else if (type == SW || type == BNEZ)
		std::cout << instructionNames[type] << " r" << src1 << " r" << src2;
	else if (type == LW)
		std::cout << instructionNames[type] << " r" << dest << " r" << src1;
	else
		std::cout << instructionNames[type] << " r" << dest << " r" << src1 << " r" << src2;
}
```

Pipeline.h

```
#include <string>
#include <vector>


/* Types of possible instruction types */
enum InstructionType {
	NOP = 0,		// NOP. Pipeline bubble.
	ADD,		// Add
	SUB,		// Subtract
	MULT,		// Multiply
	DIV,		// Divide
	LW,		// Load word
	SW,		// Store word
	BNEZ		// Branch not equal to zero
```

```
};

/* Names of possible instruction types */
const std::string instructionNames[8] = {"*", "ADD", "SUB", "MULT", "DIV", "LW", "SW", "BNEZ"};

enum Stage {
	FETCH = 0,
	DECODE,
	EXEC,
	MEM,
	WB,
	NONE
};

const std::string stageNames[6] = {"FETCH", "DECODE", "EXEC", "MEM", "WB", "NONE"};

/* A Single Register Entry containing register number and register data value */
class Register {
	public:
		Register();
		int dataValue;
		int registerNumber;
		std::string registerName;
};

/* Register file with 16 registers */
extern Register registerFile[16];

class Instruction {
	public:
		Instruction();
		Instruction(std::string);
		InstructionType type; // Type of instruction
		int dest; // Destination register number
		int src1; // Source register number
		int src2; // Source register number
		void printInstruction();
		Stage stage;
};

class Application {

	public:
		Application();
		void loadApplication(std::string);
		void printApplication();
		Instruction* getNextInstruction();
		std::vector<Instruction*> instructions;
		int PC;
};


class PipelineStage {

	public:
		PipelineStage();
		Instruction *inst;
		Stage stageType;
```

```
                void clear();
                void addInstruction(Instruction*);
                void printStage();
                void process();
};


class Pipeline {

        public:
                Pipeline(Application*);
                int cycleTime;
                Application *application;
                PipelineStage pipeline[5];
                void cycle();
                void printPipeline();
                bool done();
                bool hasDependency();
                bool forwarding;
};
```