CS-310

Lab Assignments (1-10)

Roshni Ram (201651045)
Vaaibhavi Singh (201651051)
Mahima Arora (201651055)

LAB 1

Aim

Draw 1 Bit and 4 Bit ALU with eight operations.

Software Used

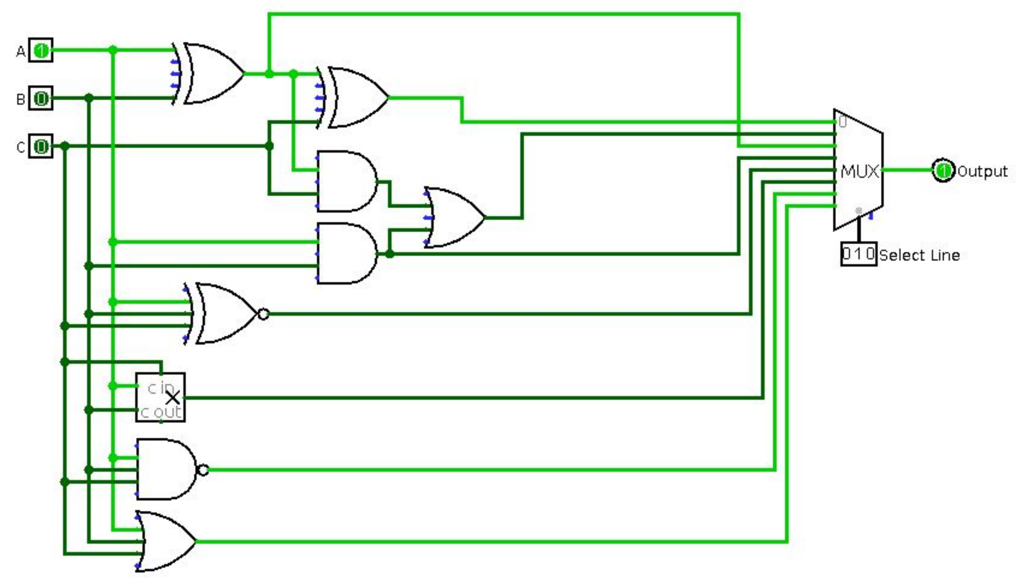
Logisim

Theory

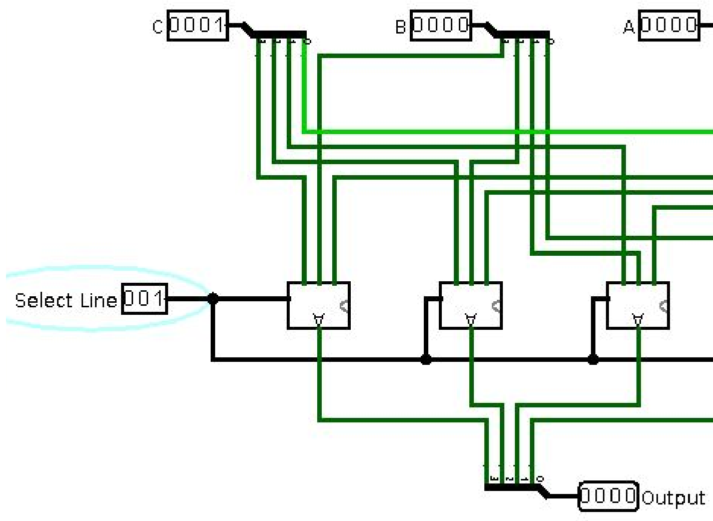
An **arithmetic logic unit (ALU)** is a digital circuit used to perform arithmetic and logical operations. It represents the fundamental building block of the **central processing unit (CPU)** of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).
Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A **register** is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

Screenshots

1-Bit ALU



4Bit ALU



Conclusion

4 Bit ALU can be made using the combination of 4 1-Bit ALUs.

LAB 2

Aim

To understand the basic principles of how pipelining works, including the problems of data and branch hazards. Finally, we should have an understanding of how the instructions are used to control different parts of the data path through a control unit. (Pipelined Processors - Single Instruction)

Questions and Answers

1. **What is a CPU?**

   A central processing unit, also called a central processor or main processor, is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output operations specified by the instructions

2. **What is an assembly language program?**

   The assembly language program is written in the low-level language which is developed by using mnemonics.

3. **How does a compiler execute simple machine language instructions?**

   It translates the given instructions into Machine code that can be executed directly by a computer's central processing unit (CPU).

4. **What is the relation between assembly language and machine language?**

   Assembly language is a more human readable view of machine language. Instead of representing the machine language as numbers, the instructions and registers are given names. Eg. for load use ld.

5. **What does the instruction ‘add t0, t1, t2’ do?**

The above instruction means: **[t0] = [t1] + [t2]**. The values are added as signed (2's complement) integers.

6. **What does the instruction 'beq t0, t1, Dest' do?**

   The above instruction can be understood as:

   If t0 == t1 then:

   go to Dest;

7. **How does the pipelined CPU different from a non-pipelined?**

   Pipelining is an implementation technique where multiple instructions are overlapped in execution whereas in non-pipelining all the actions (fetching, decoding, execution of instructions and writing the results into the memory) are grouped into a single step.

8. **Describe Hazard.**

   In the CPU design, hazards are problems with the instruction pipeline in CPU microarchitectures when the next instruction cannot execute in the following clock cycle, and can potentially lead to incorrect computation results.

Test Example

```
#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: add t0, t1, t2

nop

nop
```

```
nop

nop

.end start
```

ANSWERS

1. In this stage, the CPU reads instructions from the address in the memory whose value is present in the program counter.
2. In this stage, the instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.
3. In this stage, ALU operations are performed.
4. In this stage, memory operands are read and written from/to the memory that is present in the instruction.
5. In this stage, the computed/fetched value is written back to the register present in the instruction.
6. IF - Instruction Fetch, ID - Instruction Decode, EX - Execute, MEM - Memory, WB - Write Back. These are the five stages of RISC pipelining as explained in the above five answers.
7. 5 clock cycles required.
8. Execution Stage.
9. Memory(4th Stage) not used by arithmetic instruction as after computing it writes back to the desired location.

PROBLEM

1)#include <iregdef.h>

```
.set noreorder
```

```
.text

.global start

.ent start

start: lw t0, 0(t1)

nop

nop

nop

nop

.end start
```

a) **Stage 1** (Instruction Fetch)
In this stage, the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)
In this stage, the instruction is decoded and the register file (t0, t1) is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)
In this stage, ALU operations are performed.

R[t0] <- MEM[R[t1] + s_extend(0)];

Stage 4 (Memory Access)
In this stage, memory operands are read from the memory that is present in the instruction.

Stage 5 (Write Back)

In this stage, the computed/fetched value is written back to the register present in the instruction.

b) ALU operations are performed.

R[t0] <- MEM[R[t1] + s_extend(0)];

MEM[R[t1] + s_extend(0)] The computation of this instruction takes place in ALU in order to find the exact location from where the data is to be loaded.

c) 5 clock cycles

d) All stages used

2)

```
#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: st t0, 4(t1)

nop

nop

nop

nop

.end start
```

a) Stage 1 (Instruction Fetch)

In this stage, the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)

In this stage, the instruction is decoded and the register file (t0, t1) is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)

In this stage, ALU operations are performed.

MEM[R[t1] + sign_extend(4)] <- R[t0]

Stage 4 (Memory Access)

In this stage, memory operands are read from the memory that is present in the instruction.

b) ALU operations are performed.

MEM[R[t1] + sign_extend(4)] <- R[t0]

MEM[R[t1] + s_extend(4)] The computation of this instruction takes place in ALU in order to find the exact location where the data is to be stored.

c) 4 clock cycles

d) Write Back stage not used

3)#include <iregdef.h>

.set noreorder

.text

.global start

```
.ent start

Dest;

start: beq t0, t1, Dest

nop

nop

nop

nop

.end start
```

a) Stage 1 (Instruction Fetch)

In this stage, the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)

In this stage, the instruction is decoded and the register file (t0, t1) is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)

In this stage, ALU operations are performed.

Checks if (R[t0] == R[t1])

b) ALU checks if the value of R[t0] is equal to the value of R[t1] and if true the address of 'Dest' is passed to the Program Counter (PC).

c) 3 clock cycles

d) 3 stages used - IF, ID,EX

LAB 3

Aim

To understand the basic principles of how pipelining works, including the problems of data and branch hazards. Finally, we should have an understanding of how the instructions are used to control different parts of the data path through a control unit. (Pipelined Processors - Multiple Instructions at a Time)

Problem 1

```
#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: lui $9, 0xbf90  // load upper immediate instruction. The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes.

repeat: lbu $8, 0x0($9)   //Load Unsigned Byte

nop

sb $8, 0x0($9)  //Store the least significant byte of register Rsrc1 into memory address Rsrc2 + imm.

b repeat

nop

li $8, 0  // instruction loads a specific numeric value into that register

.end start
```

ANSWER

1. At max 4 instructions are in different stages of their execution at the same time.

Problem 2

```
#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: add t2, t0, t1

add t4, t2, t3

nop

nop

nop

.end start
```

ANSWERS

1. 4
2. 2
3. Value of t2 is required at 2nd clock cycle whereas it is available at a 4th clock cycle. This is called the data hazard (Read After Write Operation).

Problem 3

#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: lw t0, t1

add t1, t1, t0

nop

11

nop

nop

.end start

ANSWERS

1. After 4 clock cycles.
2. After 2 cycles.
3. Value of t0 is required at 2nd clock cycle whereas it is available at a 4th clock cycle. This is called the data hazard (Read After Write Operation).
4. We can use forwarding or bypassing method, basic compiler pipeline scheduling, basic dynamic scheduling, dynamic scheduling and renaming, and hardware speculation.
5. Yes, it resolves the problem as it fetches data of t0 before the 4th clock cycle.

LAB 4

Aim

To analyze the given program using pipeline simulator.

Program 1 - Addition

```
#include <iregdef.h>

.set noreorder

.text

.global start

.ent start

start: nop

nop

beq t0, t1, start

addi t0, t0, 1

nop

nop

nop

.end start
```

Questions and Answers

1. **How many cycles does it take until the branch instruction is ready to jump?**

   If the initial values of t1 and t0 are equal then five cycles for ADD, five for BEQ and 2 cycles for write back, therefore, it takes a total of 12 cycles until the branch instruction is ready to jump.

2. **What has happened with the following addi instruction while the branch is calculated?**

   While the branch value is been given to the program counter the decode stage of the ADDI instruction is completed.

3. **How does this version handle beq?**

   It keeps checking the values stored in the register t0, t1. If they are equal it jumps to start.

Program 2 - Subtraction

```
#include <iregdef.h>

.set noreorder

.text

.globl start

.ent start

start:

li t0,2

nop

nop

li t1,3
```

```
nop

nop

sub t1, t1, 1

nop

nop

nop

.end start
```

Questions and Answers

1. **How many cycles does it take until the program execution is complete?**

   It takes 12 cycles.

2. **What has happened with the following sub instruction?**

   There is a stall after instruction fetch as at the corresponding time writeback of t1 takes place because of the previous instruction.

Program 3 - Multiplication

```
#include <iregdef.h>

.set noreorder

.text

.globl start

.ent start

start:

li t0,2

nop

nop
```

```
li t1,3
nop
nop
mult  t1,t0
nop
nop
nop
.end start
```

Program 4 - Division

```
#include <iregdef.h>
.set noreorder
.text
.globl start
.ent start
start:
li t0,2
nop
nop
li t1,3
nop
nop
div  t1,t0
nop
nop
nop
.end start
```

LAB 5, 6

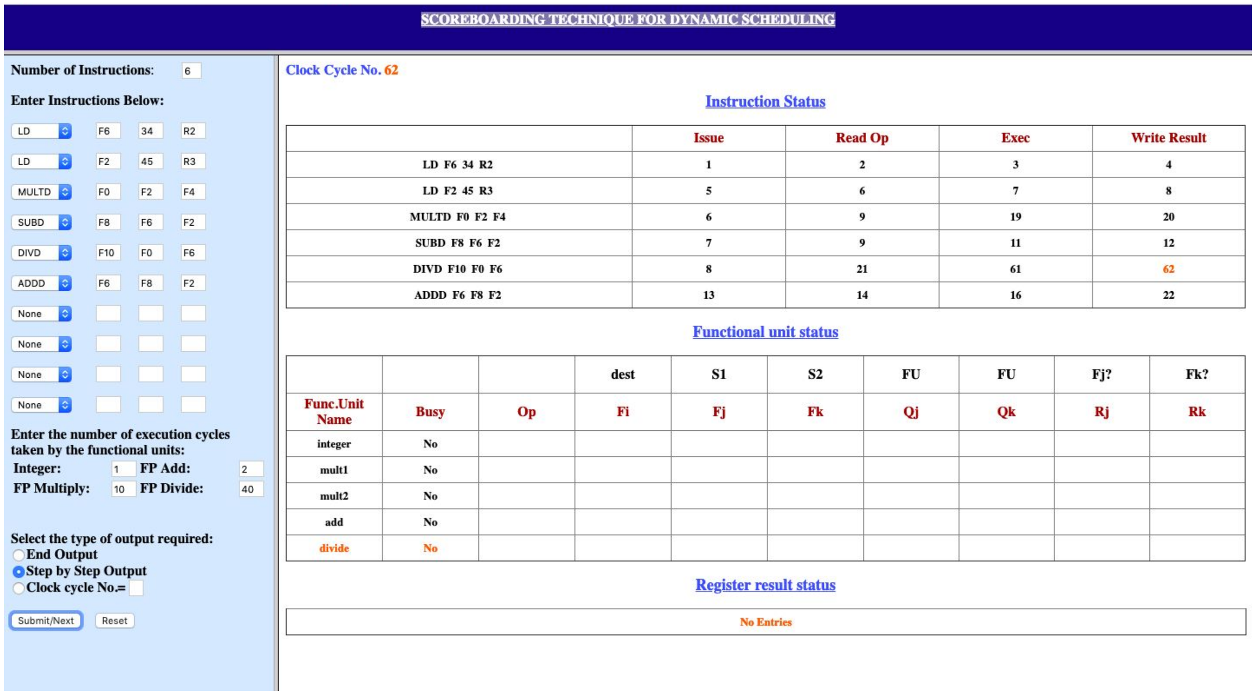
Aim

Record for each instruction in which clock-cycle it is in each of the traversed pipeline stages. Understanding the concepts of scoreboarding algorithm used for dynamic scheduling.

Experiment 1

For the first trial we look at the following program:

```
ld F6, 34(R2)
ld F2, 45(R3)
multd F0, F2, F4
subd F8, F6, F2
divd F10, F0, F6
addd F6, F8, F2
```



SCOREBOARDING TECHNIQUE FOR DYNAMIC SCHEDULING

Number of Instructions: 6

Enter Instructions Below:

LD	F6	34	R2
LD	F2	45	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:
Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Select the type of output required:
End Output [✓ Step by Step Output] Clock cycle No.=

Clock Cycle No. 62

Instruction Status

	Issue	Read Op	Exec	Write Result
LD F6 34 R2	1	2	3	4
LD F2 45 R3	5	6	7	8
MULTD F0 F2 F4	6	9	19	20
SUBD F8 F6 F2	7	9	11	12
DIVD F10 F0 F6	8	21	61	62
ADDD F6 F8 F2	13	14	16	22

Functional unit status

			dest	S1	S2	FU	FU	Fj?	Fk?
Func.Unit Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
integer	No								
mult1	No								
mult2	No								
add	No								
divide	No								

Register result status

No Entries

```
#include <iregdeg.h>
.set noreorder
.text
```

```
.global start
.ent start
start: lw t6, 34(r2)
nop
nop
lw t2, 45(r3)
nop
nop
mult t0, t2, t4
nop
nop
sub t8, t6, t2
nop
nop
div t1, t0, t6
nop
nop
add t6, t8, t2
nop
nop
.end start
```

1. After how many clock cycles can this program branch back to the beginning?
62 cycles

2. Does re-ordering influence the execution time of this program and how?

Reordering-

```
ld F2, 45(R3)
ld F6, 34(R2)
multd F0, F2, F4
subd F8, F6, F2
divd F10, F0, F6
addd F6, F8, F2
```

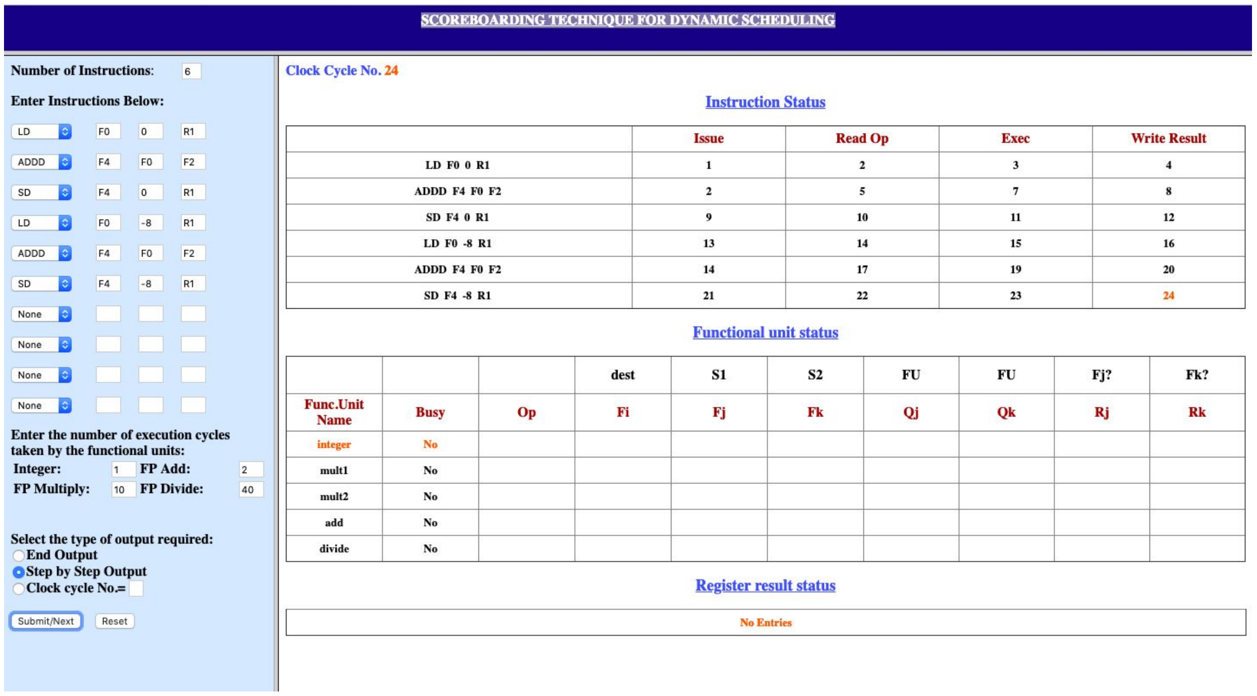
Yes. After swapping the first two instructions the number of cycles are reduced to 18.

3. Is there a Write-after-Read hazard present and how is it solved?
Yes. If we swap the first two commands, the write-after-read hazard is nullified.

Experiment 2

Another program that regularly has appeared during the lecture is the following:

```
ld F0, 0(R1)
addd F4, F0, F2
sd F4, 0(R1)
ld F0, -8(R1)
addd F4, F0, F2
sd F4, -8(R1)
```



SCOREBOARDING TECHNIQUE FOR DYNAMIC SCHEDULING

Number of Instructions: 6

Enter Instructions Below:

LD	F0	0	R1
ADDD	F4	F0	F2
SD	F4	0	R1
LD	F0	-8	R1
ADDD	F4	F0	F2
SD	F4	-8	R1
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:
Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Select the type of output required:
End Output
[✓ Step by Step Output]
Clock cycle No.=

Submit/Next Reset

Clock Cycle No. 24

Instruction Status

	Issue	Read Op	Exec	Write Result
LD F0 0 R1	1	2	3	4
ADDD F4 F0 F2	2	5	7	8
SD F4 0 R1	9	10	11	12
LD F0 -8 R1	13	14	15	16
ADDD F4 F0 F2	14	17	19	20
SD F4 -8 R1	21	22	23	24

Functional unit status

			dest	S1	S2	FU	FU	Fj?	Fk?
Func.Unit Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
integer	No								
mult1	No								
mult2	No								
add	No								
divide	No								

Register result status

No Entries

```
#include <iregdef.h>
.set reorder
.text
.global start
.ent start
start: lw t0, 0(r1)
nop
nop
add t4,t0,t2
nop
nop
sw t4, 0(r1)
nop
```

```
nop
lw t0,-8(r1)
nop
nop
add t4,t0,t2
nop
nop
sw t4, -8(r1)
nop
nop
.end start
```

1. After how many clock cycles can this program branch back to the beginning?
24 cycles

2. Does re-ordering influence the execution time of this program and how?
No, because of the presence of read-after-write and write-after-write.

3. Is there a Write-after-Read hazard present and how is it solved?
The first two instructions have write-after-read hazard and it cannot be solved as the sequence of the instructions would be disturbed.

LAB 7

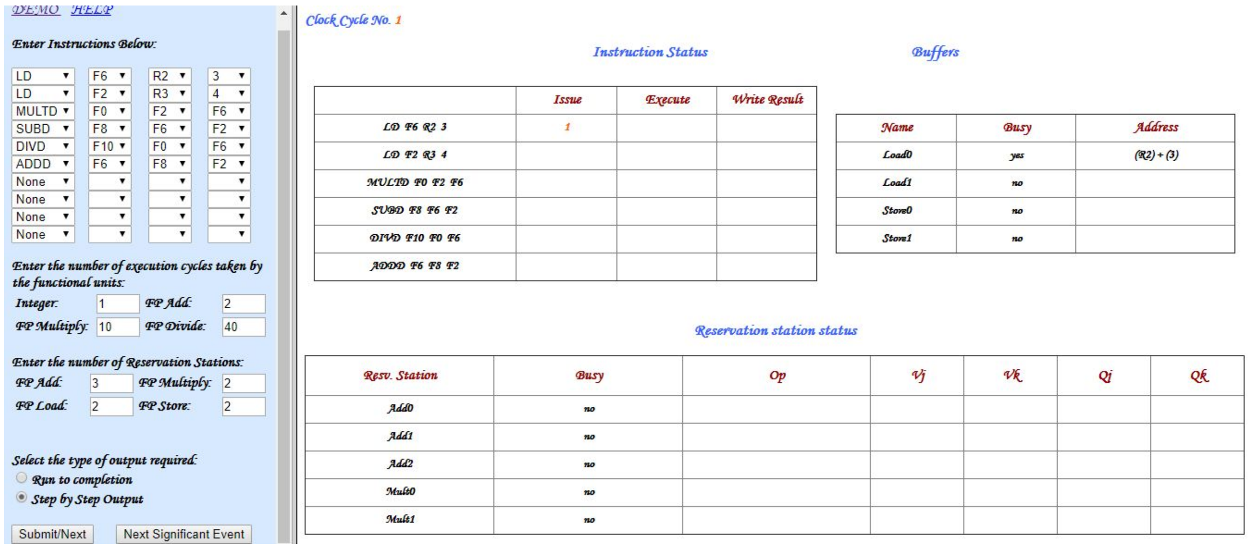
Aim

To understand Tomasulo's Algorithm for Dynamic Scheduling.

Experiment 1

For the first trial we look at the following program:

```
ld F6, 34(R2)
ld F2, 45(R3)
multd F0, F2, F6
subd F8, F6, F2
divd F10, F0, F6
addd F6, F8, F2
```



Enter Instructions Below:

LD	F6	R2	3
LD	F2	R3	4
MULTD	F0	F2	F6
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:

Run to completion [✓ Step by Step Output]

Clock Cycle No. 1

Instruction Status

	Issue	Execute	Write Result
LD F6 R2 3	1		
LD F2 R3 4			
MULTD F0 F2 F6			
SUBD F8 F6 F2			
DIVD F10 F0 F6			
ADDD F6 F8 F2			

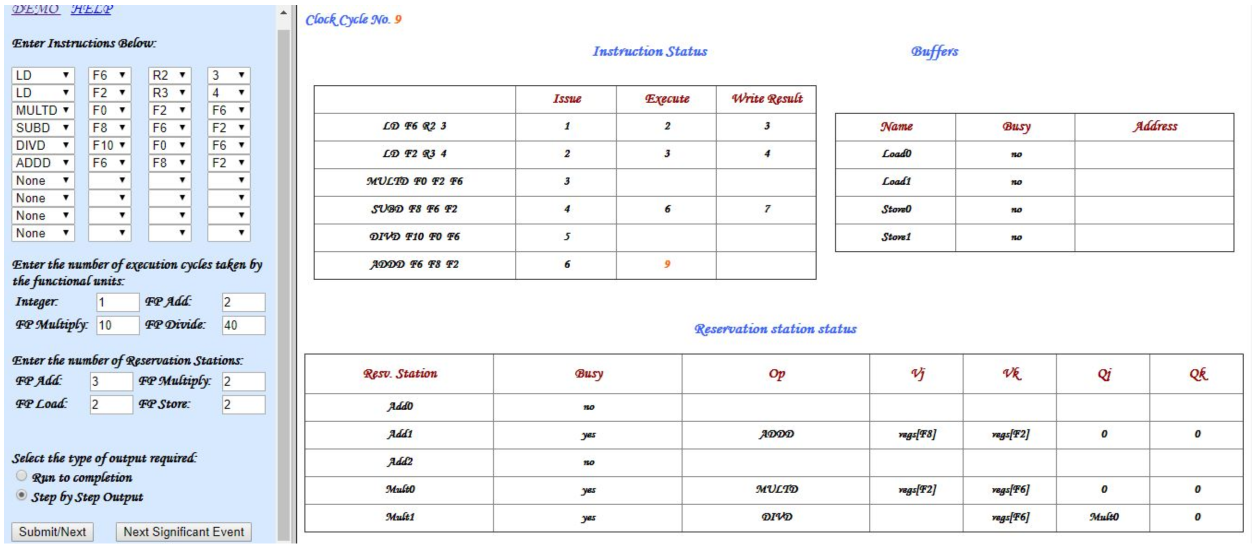
Buffers

Name	Busy	Address
Load0	yes	(R2) + (3)
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qi	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

Clock cycle 1


DEMO HELP

Enter Instructions Below:

LD	F6	R2	3
LD	F2	R3	4
MULTD	F0	F2	F6
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:
Run to completion [✓ Step by Step Output]

Submit/Next Next Significant Event

Clock Cycle No. 9

Instruction Status

	Issue	Execute	Write Result
LD F6 R2 3	1	2	3
LD F2 R3 4	2	3	4
MULTD F0 F2 F6	3		
SUBD F8 F6 F2	4	6	7
DIVD F10 F0 F6	5		
ADDD F6 F8 F2	6	9	

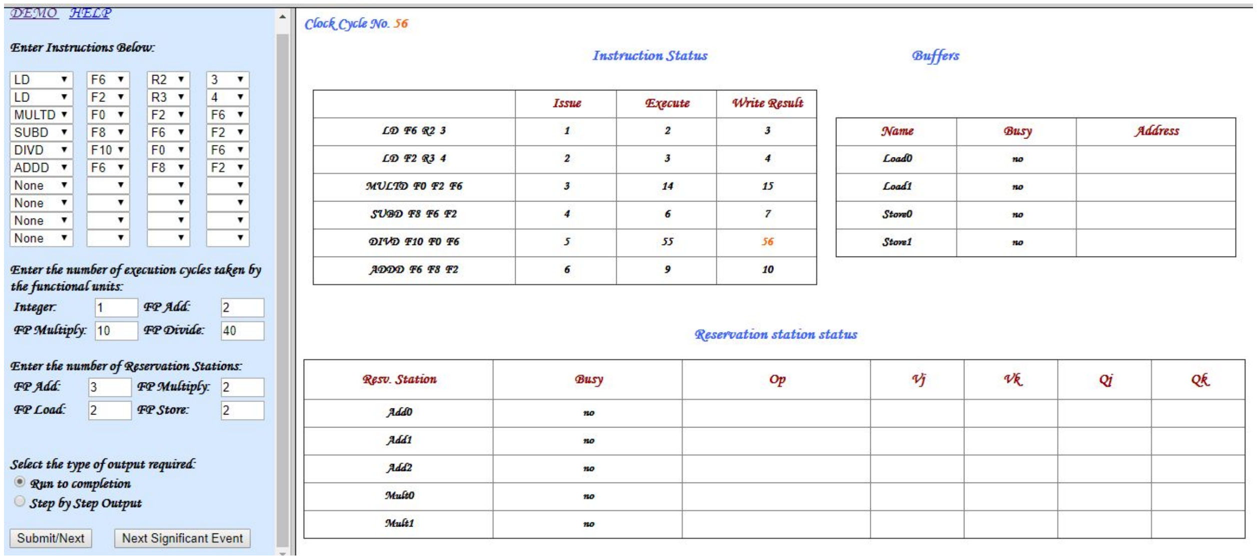
Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	yes	ADDD	regs[F8]	regs[F2]	0	0
Add2	no					
Mult0	yes	MULTD	regs[F2]	regs[F6]	0	0
Mult1	yes	DIVD		regs[F6]	Mult0	0

Clock cycle 9


DEMO HELP

Enter Instructions Below:

LD	F6	R2	3
LD	F2	R3	4
MULTD	F0	F2	F6
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:
[✓ Run to completion] Step by Step Output

Submit/Next Next Significant Event

Clock Cycle No. 56

Instruction Status

	Issue	Execute	Write Result
LD F6 R2 3	1	2	3
LD F2 R3 4	2	3	4
MULTD F0 F2 F6	3	14	15
SUBD F8 F6 F2	4	6	7
DIVD F10 F0 F6	5	55	56
ADDD F6 F8 F2	6	9	10

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

Clock cycle 56

1. After how many clock cycles can this program branch back to the beginning?
56 cycles

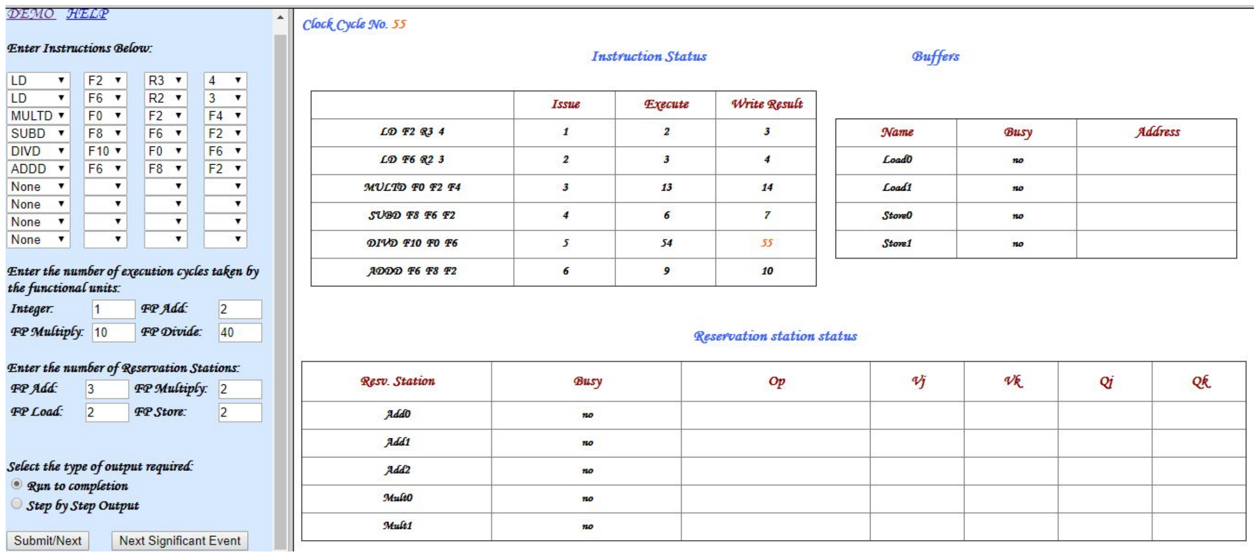
2. Does re-ordering influence the execution time of this program and how?

Reordering-

```
ld F2, 45(R3)
ld F6, 34(R2)
multd F0, F2, F4
subd F8, F6, F2
```

```
divd F10, F0, F6
addd F6, F8, F2
```

Yes. After swapping the first two instructions the number of cycles is reduced by one as the value of R6 is received first. Therefore, the total clock cycles for completion is 55.


DEMO HELP

Enter Instructions Below:

LD	F2	R3	4
LD	F6	R2	3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:
(●) Run to completion
( ) Step by Step Output

Submit/Next Next Significant Event

Clock Cycle No. 55

Instruction Status

	Issue	Execute	Write Result
LD F2 R3 4	1	2	3
LD F6 R2 3	2	3	4
MULTD F0 F2 F4	3	13	14
SUBD F8 F6 F2	4	6	7
DIVD F10 F0 F6	5	54	55
ADDD F6 F8 F2	6	9	10

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

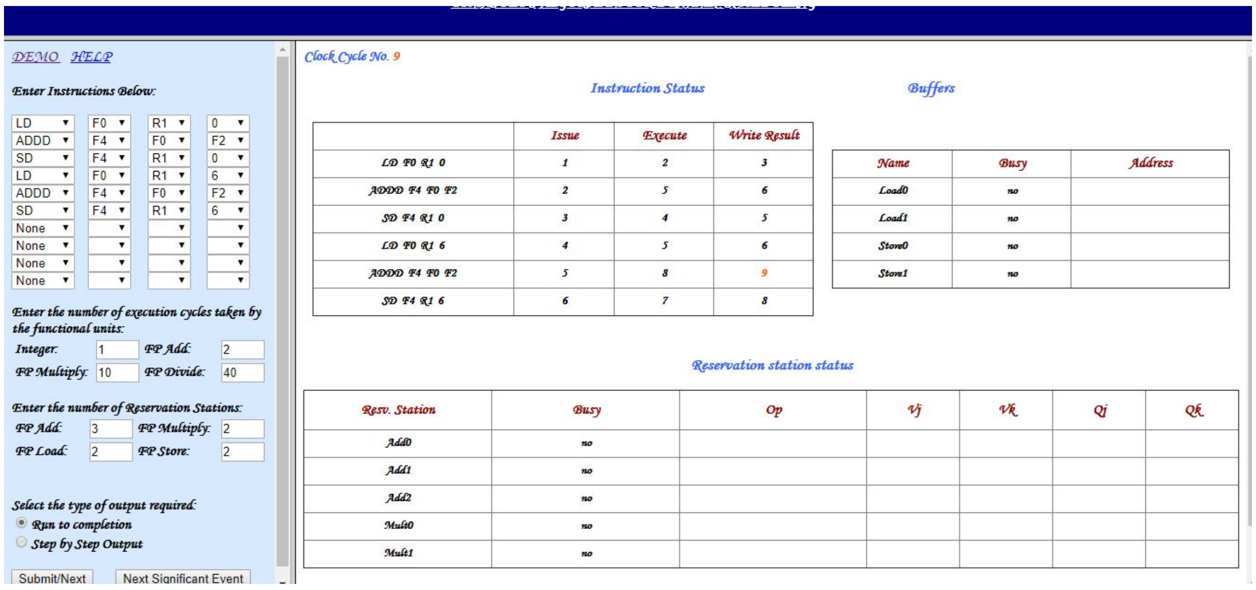
Resv. Station	Busy	Op	Vj	Vk	Qi	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

3. Is there a Write-after-Read hazard present and how is it solved?
Yes, WAR hazard is present in the first two instructions. This can be nullified by register renaming.

Experiment 2

Another program that regularly has appeared during the lecture is the following:

```
ld F0, 0(R1)
addd F4, F0, F2
sd F4, 0(R1)
ld F0, -8(R1)
addd F4, F0, F2
sd F4, -8(R1)
```



Enter Instructions Below:

LD	F0	R1	0
ADDD	F4	F0	F2
SD	F4	R1	0
LD	F0	R1	6
ADDD	F4	F0	F2
SD	F4	R1	6
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:
[✓ Run to completion]
Step by Step Output

Clock Cycle No. 9

Instruction Status

	Issue	Execute	Write Result
LD F0 R1 0	1	2	3
ADDD F4 F0 F2	2	5	6
SD F4 R1 0	3	4	5
LD F0 R1 6	4	5	6
ADDD F4 F0 F2	5	8	9
SD F4 R1 6	6	7	8

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

1. After how many clock cycles can this program branch back to the beginning?
9 cycles

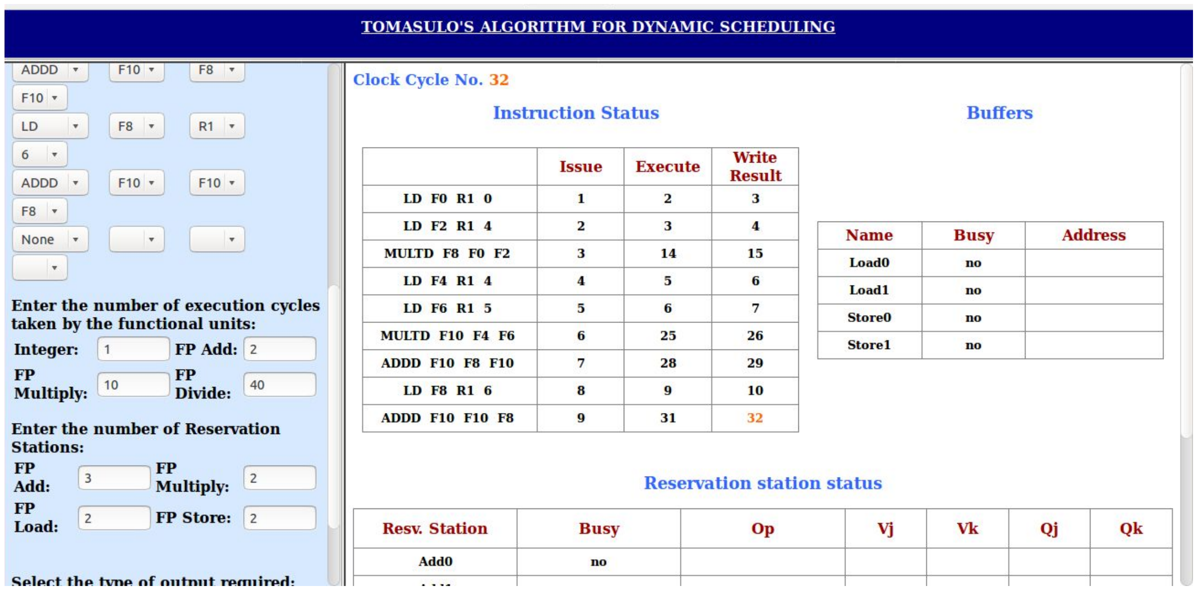
2. Does re-ordering influence the execution time of this program and how?
No.

3. Is there a Write-after-Read hazard present and how is it solved?
There is no presence of WAR hazard.

Experiment 3

The last program that we will look at is the sum-of-products that appears in the Fast Fourier transform.

```
ld F0, 0(R1)
ld F2, 4(R1)
multd F8, F0, F2
ld F4, 8(R1)
ld F6, 10(R1)
multd F10, F4, F6
addd F10, F8, F10
ld F8, 12(R1)
addd F10, F10, F8
```



TOMASULO'S ALGORITHM FOR DYNAMIC SCHEDULING

ADDD F10 F8
F10
LD F8 R1
6
ADDD F10 F10
F8
None

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Enter the number of Reservation Stations:

FP Add: 3 FP Multiply: 2
FP Load: 2 FP Store: 2

Select the type of output required:

Clock Cycle No. 32

Instruction Status

	Issue	Execute	Write Result
LD F0 R1 0	1	2	3
LD F2 R1 4	2	3	4
MULTD F8 F0 F2	3	14	15
LD F4 R1 4	4	5	6
LD F6 R1 5	5	6	7
MULTD F10 F4 F6	6	25	26
ADDD F10 F8 F10	7	28	29
LD F8 R1 6	8	9	10
ADDD F10 F10 F8	9	31	32

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					

1. After how many clock cycles can this program branch back to the beginning?
32 cycles

2. Does re-ordering influence the execution time of this program and how?
No.

3. Is there a Write-after-Read hazard present and how is it solved?
Yes. WAR hazard is present between the 7th and the 8th instruction. This can be resolved by register renaming.

Question 4

Differences between Scoreboarding and Tomasulo’s Algorithm.

Tomasulo’s Algorithm	**Scoreboard Algorithm**
Pipelined functional units	Multiple functional units
No issues of structural hazards due to the availability of multiple functional units	Stalls may occur due to the unavailability of functional units
WAR and WAW hazards can be avoided using register renaming	WAR and WAW can be overcome by introducing stalls
Operand values are taken from reservation stations	Operand values are taken from registers
Results are passed to FUs from reservation stations which are similar to forwarding	Results are passed through registers
Hazard detection and execution, control functionality is distributed	Hazard detection and execution, control functionality is entirely taken care of by scoreboard

LAB 8

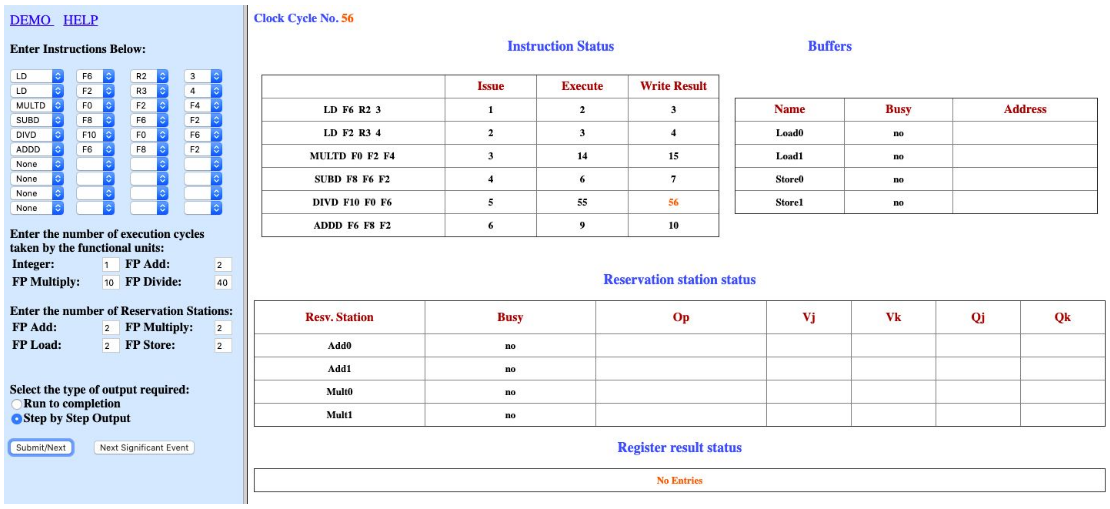
Aim

Scheduling the given codes using Tomasulo's algorithm and Score Board algorithms.

Question 1:

```
ld F6, 34(R2)
ld F2, 34(R3)
multd F0, F2, F4
subd F8, F6, F2
divd F10, F0, F6
addd F6, F8, F2
```

Using Tomasulo's algorithm



Enter Instructions Below:

LD	F6	R2	3
LD	F2	R3	4
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer:	1	FP Add:	2
FP Multiply:	10	FP Divide:	40

Enter the number of Reservation Stations:

FP Add:	2	FP Multiply:	2
FP Load:	2	FP Store:	2

Select the type of output required:
Run to completion [✓ Step by Step Output]

Clock Cycle No. 56

Instruction Status

	Issue	Execute	Write Result
LD F6 R2 3	1	2	3
LD F2 R3 4	2	3	4
MULTD F0 F2 F4	3	14	15
SUBD F8 F6 F2	4	6	7
DIVD F10 F0 F6	5	55	56
ADDD F6 F8 F2	6	9	10

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

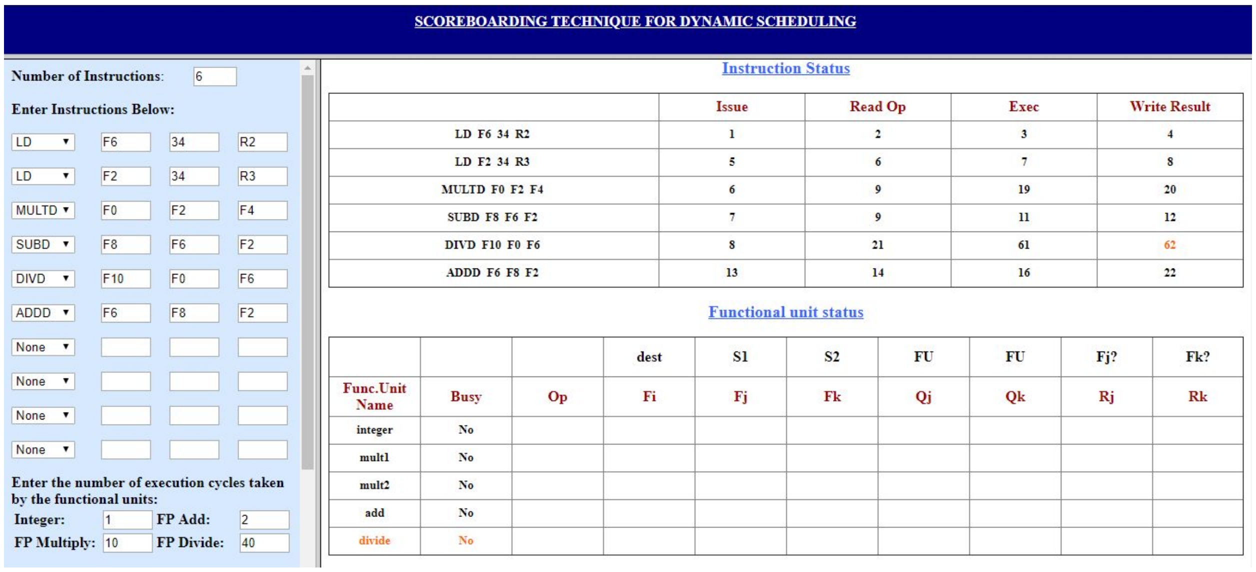
Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	no					
Mult0	no					
Mult1	no					

Register result status

No Entries

Using Scoreboard's algorithm



SCOREBOARDING TECHNIQUE FOR DYNAMIC SCHEDULING

Number of Instructions: 6

Enter Instructions Below:

LD	F6	34	R2
LD	F2	34	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer: 1 FP Add: 2
FP Multiply: 10 FP Divide: 40

Instruction Status

	Issue	Read Op	Exec	Write Result
LD F6 34 R2	1	2	3	4
LD F2 34 R3	5	6	7	8
MULTD F0 F2 F4	6	9	19	20
SUBD F8 F6 F2	7	9	11	12
DIVD F10 F0 F6	8	21	61	62
ADDD F6 F8 F2	13	14	16	22

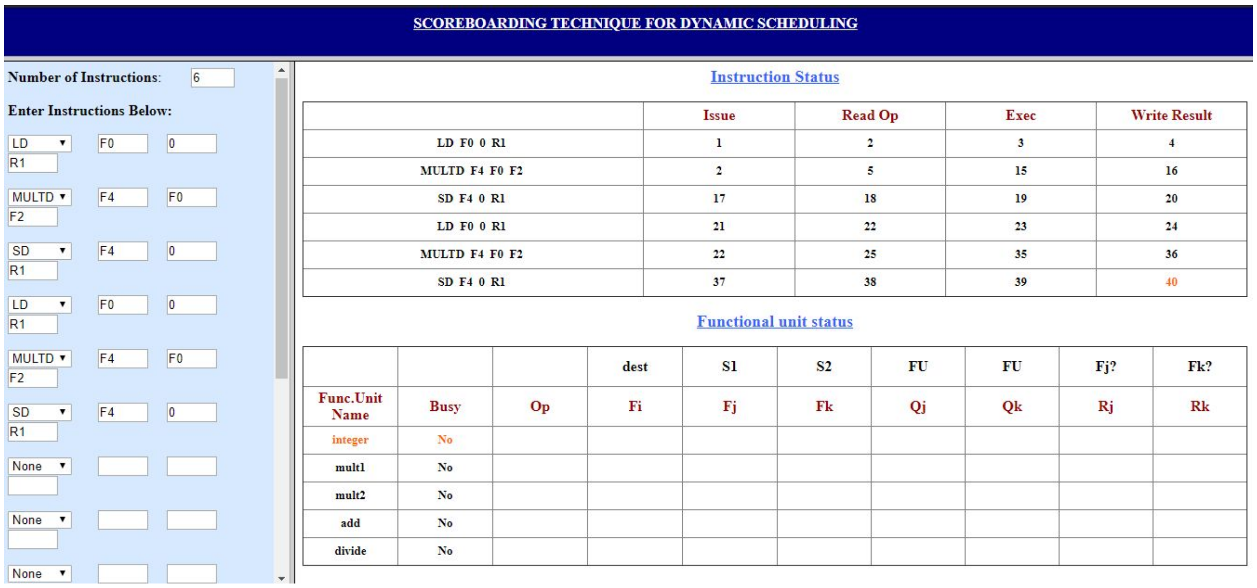
Functional unit status

			dest	S1	S2	FU	FU	Fj?	Fk?
Func.Unit Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
integer	No								
mult1	No								
mult2	No								
add	No								
divide	No								

1. In which clock cycle (numbered 0,1,2,...) does the second LD instruction complete?
   Ans. 5
2. In which clock cycle does the MULTD instruction complete?
   Ans. 16
3. In which clock cycle does the ADDD instruction complete?
   Ans. 11

Question 2

Using Scoreboard's algorithm



SCOREBOARDING TECHNIQUE FOR DYNAMIC SCHEDULING

Number of Instructions: 6

Enter Instructions Below:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
None			
None			
None			

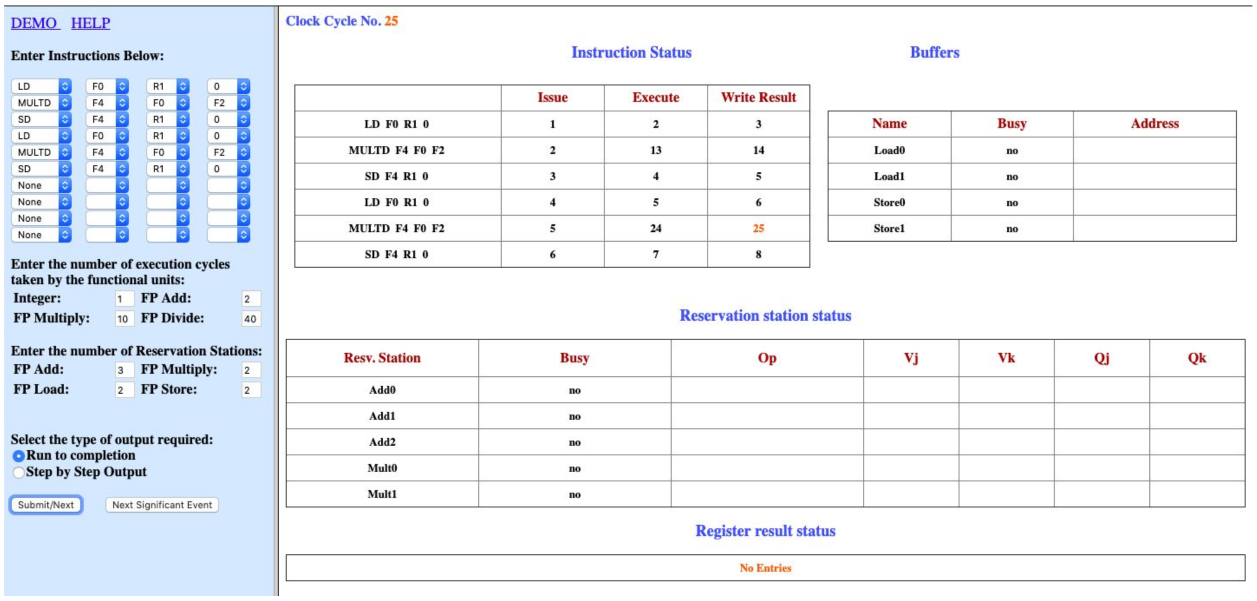
Instruction Status

	Issue	Read Op	Exec	Write Result
LD F0 0 R1	1	2	3	4
MULTD F4 F0 F2	2	5	15	16
SD F4 0 R1	17	18	19	20
LD F0 0 R1	21	22	23	24
MULTD F4 F0 F2	22	25	35	36
SD F4 0 R1	37	38	39	40

Functional unit status

			dest	S1	S2	FU	FU	Fj?	Fk?
Func.Unit Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
integer	No								
mult1	No								
mult2	No								
add	No								
divide	No								

Using Tomasulo's algorithm


DEMO HELP

Enter Instructions Below:

LD	F0	R1	0
MULTD	F4	F0	F2
SD	F4	R1	0
LD	F0	R1	0
MULTD	F4	F0	F2
SD	F4	R1	0
None			
None			
None			
None			

Enter the number of execution cycles taken by the functional units:

Integer:	1	FP Add:	2
FP Multiply:	10	FP Divide:	40

Enter the number of Reservation Stations:

FP Add:	3	FP Multiply:	2
FP Load:	2	FP Store:	2

Select the type of output required:
[✓ Run to completion]
Step by Step Output

Submit/Next   Next Significant Event

Clock Cycle No. 25

Instruction Status

	Issue	Execute	Write Result
LD F0 R1 0	1	2	3
MULTD F4 F0 F2	2	13	14
SD F4 R1 0	3	4	5
LD F0 R1 0	4	5	6
MULTD F4 F0 F2	5	24	25
SD F4 R1 0	6	7	8

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

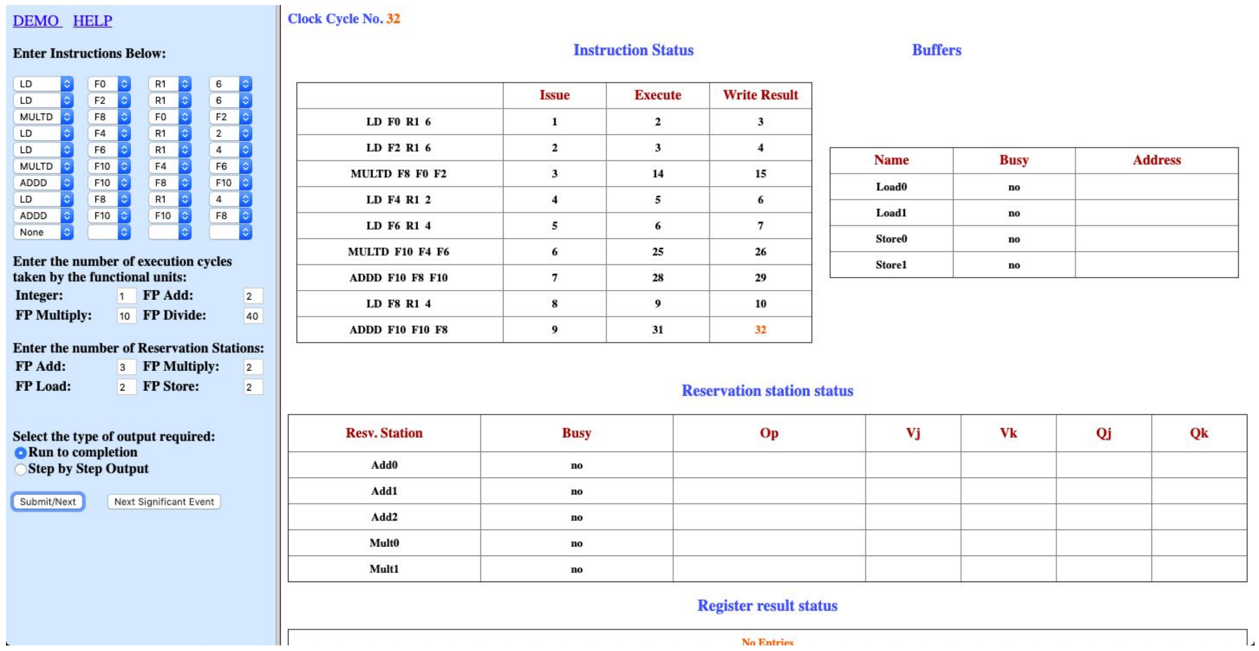
Register result status

No Entries

1. In which clock cycle does the second SD instruction complete?
   Ans. 9
2. In which clock cycle does the first MULTD instruction complete
   Ans. 15
3. In which clock cycle does the second MULTD instruction complete?
   Ans. 20

Question 3

Using Tomasulo’s algorithm



DEMO HELP

Enter Instructions Below:

LD	F0	R1	6
LD	F2	R1	6
MULTD	F8	F0	F2
LD	F4	R1	2
LD	F6	R1	4
MULTD	F10	F4	F6
ADDD	F10	F8	F10
LD	F8	R1	4
ADDD	F10	F10	F8
None			

Enter the number of execution cycles taken by the functional units:

Integer:	1	FP Add:	2
FP Multiply:	10	FP Divide:	40

Enter the number of Reservation Stations:

FP Add:	3	FP Multiply:	2
FP Load:	2	FP Store:	2

Select the type of output required:
[✓ Run to completion]
Step by Step Output

Clock Cycle No. 32

Instruction Status

	Issue	Execute	Write Result
LD F0 R1 6	1	2	3
LD F2 R1 6	2	3	4
MULTD F8 F0 F2	3	14	15
LD F4 R1 2	4	5	6
LD F6 R1 4	5	6	7
MULTD F10 F4 F6	6	25	26
ADDD F10 F8 F10	7	28	29
LD F8 R1 4	8	9	10
ADDD F10 F10 F8	9	31	32

Buffers

Name	Busy	Address
Load0	no	
Load1	no	
Store0	no	
Store1	no	

Reservation station status

Resv. Station	Busy	Op	Vj	Vk	Qj	Qk
Add0	no					
Add1	no					
Add2	no					
Mult0	no					
Mult1	no					

Register result status

No Entries

1. After how many clock cycles can this program branch back to the beginning?
   Ans. 33
2. Does re-ordering influence the execution time of this program and how?
   Ans. Yes
3. Is there a Write-after-Read hazard present and how is it solved?
   Ans. Yes

```
ADDD F10, F8, F10
LD F8, 10(R1)
```

LAB 9

Aim

Understand how a program behaves related to branch prediction.

Question 1

1. **What is the role of simulators in processor design?**
   Ans. When computer architecture researchers work to improve the performance of a computer system, they often use an existing system to simulate a proposed system. Performance estimation is one of the most important results obtained by simulation.

2. **Why is it advantageous to have several different simulators?**
   Ans. The more state that is simulated, the longer a simulation will take. Complex simulations can be executed 100s of times slower than a real processor. Therefore, simulating the execution of a program that would take an hour of CPU time on an existing processor can take a week on a complex simulator. For this reason, it is important to evaluate what measurements are desired and limit the simulation to only the state that is necessary to properly estimate those measurements. This is the reason for the inclusion of several different simulators in the SimpleScalar tool set.

3. **For the four branch prediction schemes 'taken | perfect | bimod | comb ', describe the predictor. Your description should include:**
   a. **What information the predictor stores (if any)?**
   b. **How the prediction is made?**

   Ans. Branch predictors predict the next fetch address (to be used in the next cycle).
   Taken - predict the branch taken if branched the last time.
   Not Taken - predict the branch not-taken, if didn't branch the last time
   Bimod - It is a simple predictor, which uses 2-bit saturating counters to predict if a given branch is likely to be taken or not.
   Comb - A hybrid predictor, implements more than one prediction mechanism. The final prediction is based either on a meta-predictor which remembers which of the predictors has made the best prediction in the past or a majority vote function based on an odd number of different predictors.

4. **What is out-of-order execution?**
   Ans. This simulator implements a very detailed out-of-order issue superscalar processor with a two-level memory system and speculative execution support. This simulator is a performance simulator, tracking the latency of all pipeline operations.

5. **What is the difference between scoreboarding and Tomasulo?**

Tomasulo's Algorithm	Scoreboarding Algorithm
Pipelined functional units	Multiple functional units
No issues of structural hazards due to the availability of multiple functional units	Stalls may occur due to the unavailability of functional units
WAR and WAW hazards can be avoided using register renaming	WAR and WAW can be overcome by introducing stalls
Operand values are taken from reservation stations	Operand values are taken from registers
Results are passed to FUs from reservation stations which are similar to forwarding	Results are passed through registers
Hazard detection and execution, control functionality is distributed	Hazard detection and execution, control functionality is entirely taken care of by scoreboard

Experiment 1

Benchmark programs versus instruction class profiles.

Benchmark	Load	Store	Uncond branch	Cond branch	Integer Computation	Fp computation
anagram	23.75	9.66	5.82	18.45	42.30	0.00
go	19.67	6.30	3.18	12.74	58.11	0.00
compress	5.58	59.20	1.14	747	25.02	1.59
applu	20.88	5.16	0.27	3.95	48.26	21.47
mgrid	0.18	14.17	0.02	14.37	71.26	0.00
swim	24.09	6.92	2.57	3.46	38.54	24.42
perl	26.30	17.88	5.80	12.68	37.09	0.25

gcc	25.80	13.38	4.81	15.51	40.49	0.01

Table 1 (Benchmark programs versus instruction class profiles in %)

Questions:

1. **Is your benchmark memory intensive or computation intensive?**

   Ans. Benchmark is more memory intensive.

2. **Is your benchmark mainly using integer or floating point?**

   Ans. Equally using both.

3. **What percentage of the instructions executed are conditional branches? Given this percentage, how many instructions on average does the processor execute between each pair of conditional branch instructions (do not include the conditional branch instructions).**

   Ans. Approximately 25% instructions executed are conditional branches. On an average, 3 instructions are executed between each pair of conditional branches.

4. **Using your textbook, class notes, other references, and your own opinion, list and explain several reasons why the performance of a processor like the one simulated by sim-outorder (e.g., out-of-order-issue superscalar) will suffer because of conditional branches. For each reason also explain how, if at all, a branch predictor could help the situation.**

   Ans. It simulates everything that happens in a superscalar processor pipeline, including out-of-order instruction issue, the latency of the different execution units, the effects of using a branch predictor, etc. Because of this, sim-outorder runs more slowly, but it also generates much more information about what happens in a processor. Yes, branch predictor could help in this situation as it would increase the efficiency by predicting which branch should be taken.

Experiment 2

Branch prediction statistics for the three branch prediction schemes, **'nottaken | taken | bimod'**

Benchmark	**Not Taken**	**Taken**	**Bimod**
anagram	0.5920	0.3520	0.9539
go	0.5184	0.3214	0.8432
compress	0.2492	0.1172	0.9742
applu	0.4005	0.3375	0.8050
mgrid	0.0182	0.017	0.9921
swim	0.5292	0.8977	0.9821
perl	0.6476	0.3339	0.9538
gcc	0.5994	0.367	0.8931

Table 2 (Branch prediction statistics)

LAB 10

Experiment 1

1. **Branch prediction Rate versus CPI**

	Taken		**Bimod**		**Comb**		**Perfect**	
Benchmark	Branch Pred. Rate	CPI	Branch Pred. Rate	CPI	Branch Pred. Rate	CPI	Branch Pred. Rate	CPI
anagram	0.3520	2.0973	0.9539	1.4083	0.9721	1.3879		1.3492
go	0.3214	2.4698	0.8432	2.1000	0.8469	2.0913		1.9520
compress	0.1172	2.091	0.9742	1.4803	0.9790	1.7215		1.7102
applu	0.3375	1.4313	0.8050	1.3580	0.9334	1.3371		1.3250
mgrid	0.017	1.8553	0.9921	1.1543	0.9921	1.1543		1.1463
perl	0.3339	2.3327	0.9538	1.7198	0.9846	1.6851		1.6196
gcc	0.367	3.4579	0.8931	2.8567	0.9116	2.8358		2.7015

Table 3 (Branch prediction Rate versus CPI)

Questions:

For the four branch prediction schemes ' **taken|perfect|bimod|comb** ', describe the predictor.

Your description should include:

1. **What information the predictor stores (if any)?**

   Ans. It stores the number of instructions/ load/ stores executed, total simulated time in seconds, simulation speed, instructions per branch, total number of branches, branch prediction rate and cpi.

2. **How the prediction is made?**

Ans. **Taken** - predict the branch taken if branched the last time.

Not Taken - predict the branch not-taken, if didn't branch the last time

Bimod - It is a simple predictor, which uses 2-bit saturating counters to predict if a given branch is likely to be taken or not.

Comb - A hybrid predictor, implements more than one prediction mechanism. The final prediction is based either on a meta-predictor which remembers which of the predictors has made the best prediction in the past or a majority vote function based on an odd number of different predictors.

3. **What the relative accuracy of the predictor is compared to the others.**

   Ans. Comb 95%, Bimod 90% and Taken 30%.

Experiment 2

Choosing a new branch strategy

Benchmark		**Taken**	**Bimod**	**Comb**	**Perfect**
anagram	Sim_cycle	39008444	26194143	25814329	25094502
	CPI	2.0973	1.4083	1.3879	1.3492
	Exe time (secs)	6	6	7	6
go	Sim_cycle	77510743	65903958	65630686	61258936
	CPI	2.4698	2.1000	2.0913	1.9520
	Exe time(secs)	11	12	12	10
applu	Sim_cycle	51308990	48682105	47932800	47498378
	CPI	1.4313	1.3580	1.3371	1.3250
	Exe Time(secs)	12	11	12	11

Table 4 (Impact of different branch strategies)

Questions:

1. **What would be your choice for your benchmark? Why?**

   Ans. Anagram as execution time and are comparatively less. Throughput is better.

2. **How much do you have to be able to increase the clock frequency in order to gain performance when allowing a branch miss-prediction latency of 3 cycles instead of 2 when using the taken predictor?**

   Ans. In anagram cpi decreased from 2.09 to 1.94 i.e. there is an increase in clock frequency, when allowing a branch misprediction latency of 3 cycles instead of 2.

In-order vs out-of-order issue

Experiment 3

Benchmark	**taken**	**Pipeline width = 1**		**Pipeline width = 4**		**Pipeline width = 8**	
		Sim_cycle	**CPI**	**Sim_cycle**	**CPI**	**Sim_cycle**	**CPI**
anagram	**In-order**	40105147	2.1562	36841666	1.9808	36818248	1.9795
	Out-of-order	3900844	2.0973	26555414	1.4277	26123216	1.4045
go	**In-order**	81371767	2.5928	71859618	2.2897	70248712	2.2384
	Out-of-order	77510743	2.4698	57437650	1.8302	55660896	1.7736
applu	**In-order**	56835033	1.5854	53028912	1.4793	52909652	1.4759
	Out-of-order	51308990	1.4313	27727080	0.7735	27127743	0.7567

Table 5 (In-order and out-of-order issue versus pipeline width)

Questions:

1. **What is the impact on CPI of the increased pipeline width?**

Ans. Cpi decreases with increase in pipeline width.

2. **Explain the impact and their difference for both in-order and out-of-order issue.**

   Ans. In out-of-order CPI and simulation cycles is less as compared to in-order issue. In both in-order and out-of-order with increase in pipeline width the attribute simulation cycle as well as CPI decreases.

Experiment 1.2

Benchmark	taken	Memory Port = 1		Memory Port = 4		Memory Port = 8	
		Sim_cycle	**CPI**	**Sim_cycle**	**CPI**	**Sim_cycle**	**CPI**
anagram	**In-order**	40105147	2.1562	40105147	2.1562	40105147	2.1562
	Out-of-order	39008689	2.0973	39008444	2.0973	39008444	2.0973
go	**In-order**	81371767	2.5928	81371767	2.5928	81371767	2.5928
	Out-of-order	77510734	2.4698	77510743	2.4698	77510743	2.4698
applu	**In-order**	56838043	1.5855	56835033	1.5854	56835033	1.5854
	Out-of-order	51317511	1.4315	51308990	1.4313	51308990	1.4313

Table 5 (In-order and out-of-order issue versus pipeline width)

Questions:

1. **What is the impact on CPI of the increase in available memory ports?**

   Ans. CPI remains the same in case of increase in the available memory ports.

2. **Why is bimodal branch prediction more expensive to implement than predict not taken?**

   Ans. Bimod uses 2-bit saturating counters to predict if a given branch is likely to be taken or not and not taken predicts the branch as not-taken, if didn't branch the last time. Therefore bimod is more expensive than not taken as computations are more.

3. **Why is bimodal better than not taken?**

   Ans. Bimod uses 2-bit saturating counters to predict if a given branch is likely to be taken or not and not taken predicts the branch as not-taken, if didn't branch the last time.

4. **What, if any, is the impact on CPI by allowing out-of-order issue?**

   Ans. CPI is less in case of out-of-order issue as compared to in-order issue.

5. **What, if any, is the impact on CPI by allowing more instructions to be processed in one cycle?**

   Ans. CPI decreases.

6. **Is the wider pipeline more effective with an in-order or out-of-order issue, and if so - why?**

   Ans. If we increase the pipeline width, the instructions per cycle increases, so CPI decreases. So, it is more effective for in-order as it has greater CPI as compared to out-of-order issue.