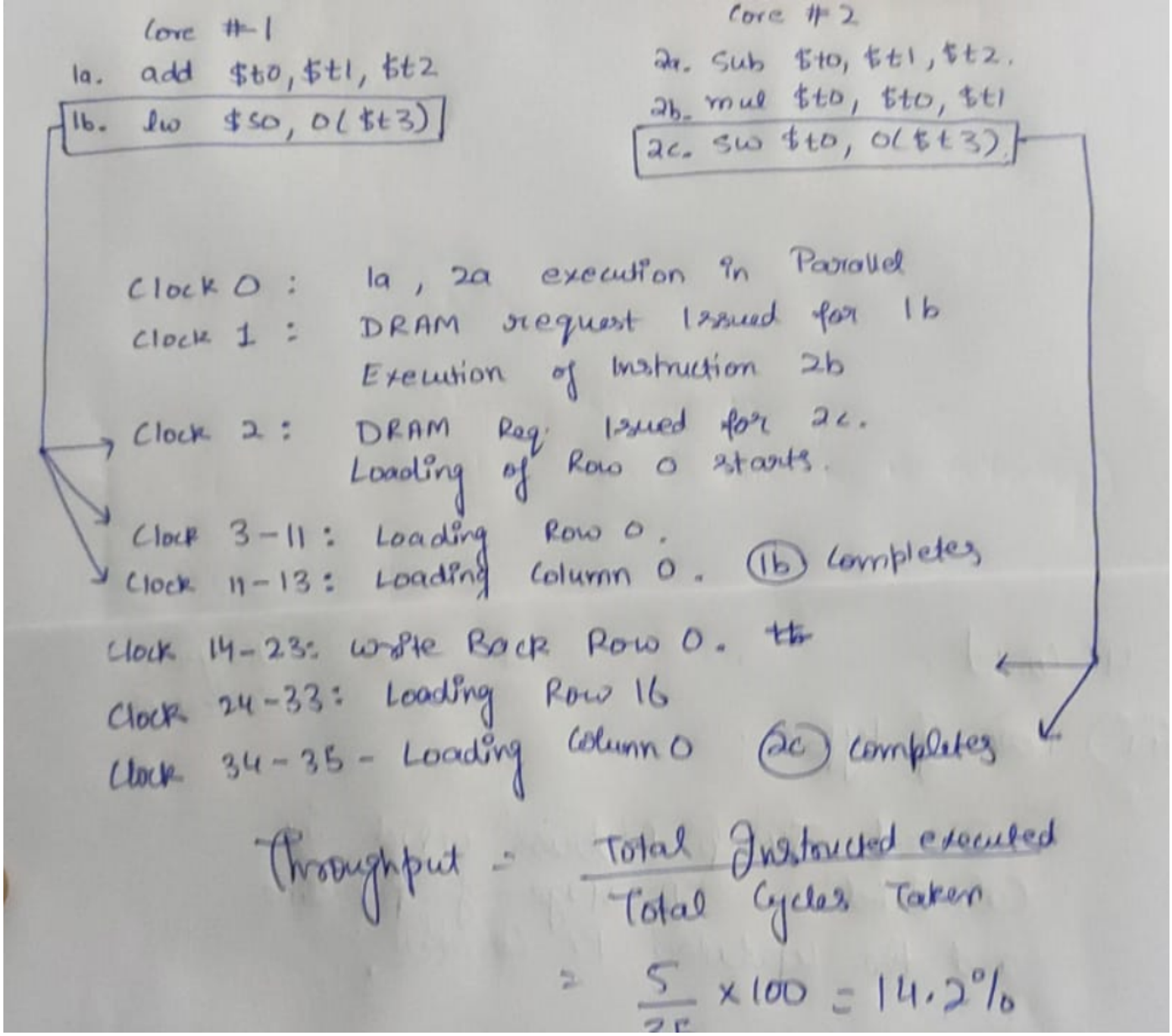
Testing:

Anirudha Kulkarni : 2019CS50421 Pratyush Saini : 2019CS10444

1. Verification of example given in Problem statement with respect to our implementation

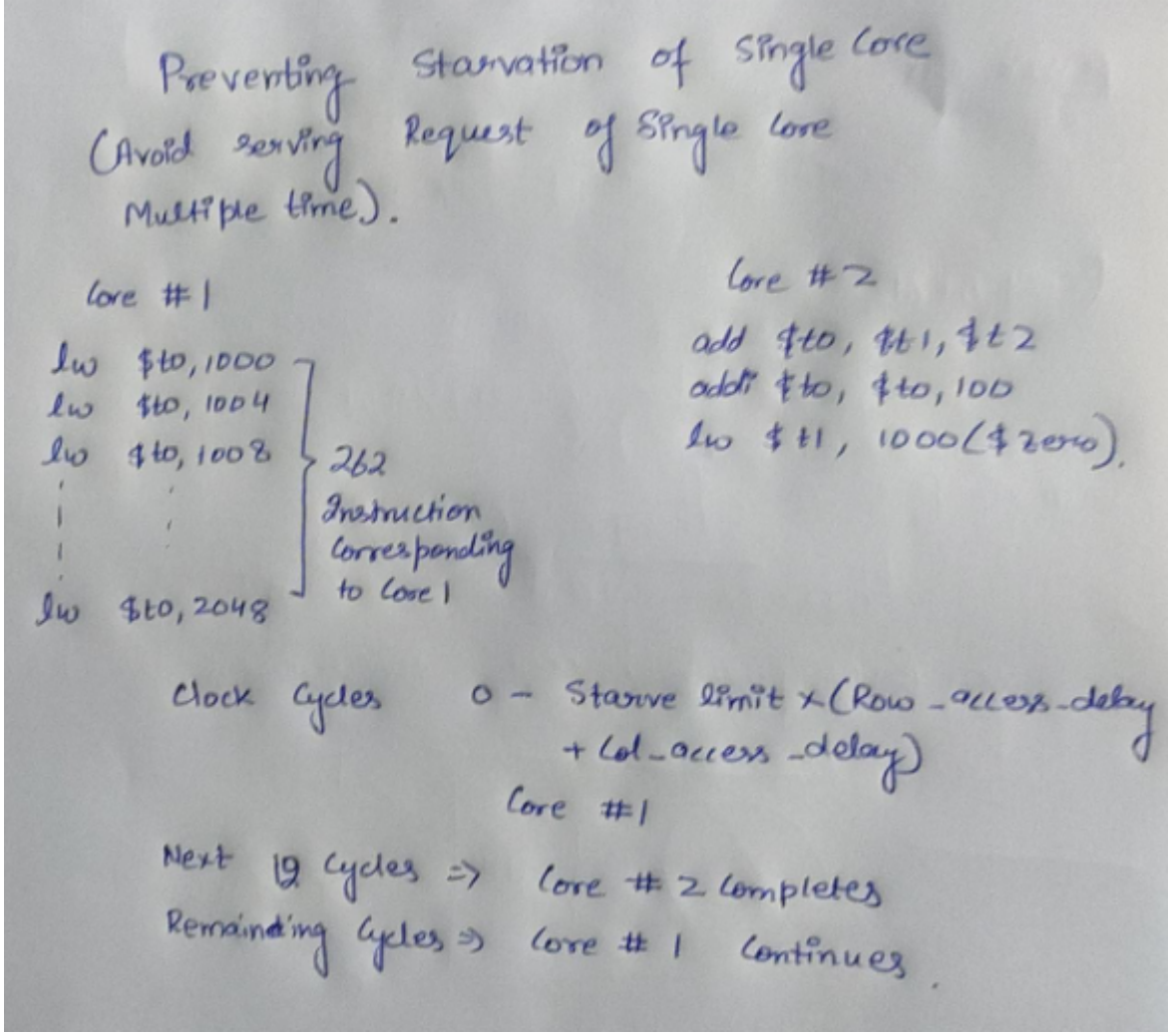
```
Total Instructions executed in 35 cycles = 5
Average Number of Instructions per cycle (IPC) = 0.142857
Throughput Efficiency 14.2857 %


Program Execution successful

============================================================
Number of times each instruction was executed:

Core #1:
add $t0, $t1, $t2
1
lw $s0, 0($t3)
1

Core #2:
sub $t0, $t1, $t2
1
mul $t0, $t0, $t1
1
sw $t0, 0($t3)
1


============================================================
Estimated Delay caused due to MRM Operations = 0 clock cycles.
============================================================
Memory content at the end of the execution

Core #1:

Core #2:
0-3: 0
```

Explaination:



2. Multi core starving:

There can be a case when a core never leaves the MRM and keeps on sending request to MRM due to algorithmic bias. Hence we have starving limit on each core after which the core must get lower priority and release the MRM.



Here we have a lot of DRM requests from Core #1. And as the core #1 is giving algorithmic benefit of lesser clock cycles core #2 was likely to be starved. But by handeling starvation in we see change in cycle 153 to 154 that Row 1 is loaded into row buffer which corresponds to Core #2

```
Clock cycle 131
Core #1:
Core #2:
Writing Back Row 1 in Row Buffer

Clock cycle 132
Core #1:
Core #2:
Loading Row 17 in Row Buffer
```

Again after resolving starving we change back to algorithmically most beneficial request that is corresponding to core #1.

```
Clock cycle 153
Core #1:
Core #2:
Writing Back Row 17 in Row Buffer

Clock cycle 154
Core #1:
Core #2:
Loading Row 1 in Row Buffer
```

Hence finally :

```
Total Instructions executed in 226 cycles = 55
Average Number of Instructions per cycle (IPC) = 0.243363
Throughput Efficiency 24.3363 %


Program Execution successful
```

```
============================================================
Estimated Delay caused due to MRM Operations = 50 clock cycles.
============================================================
```

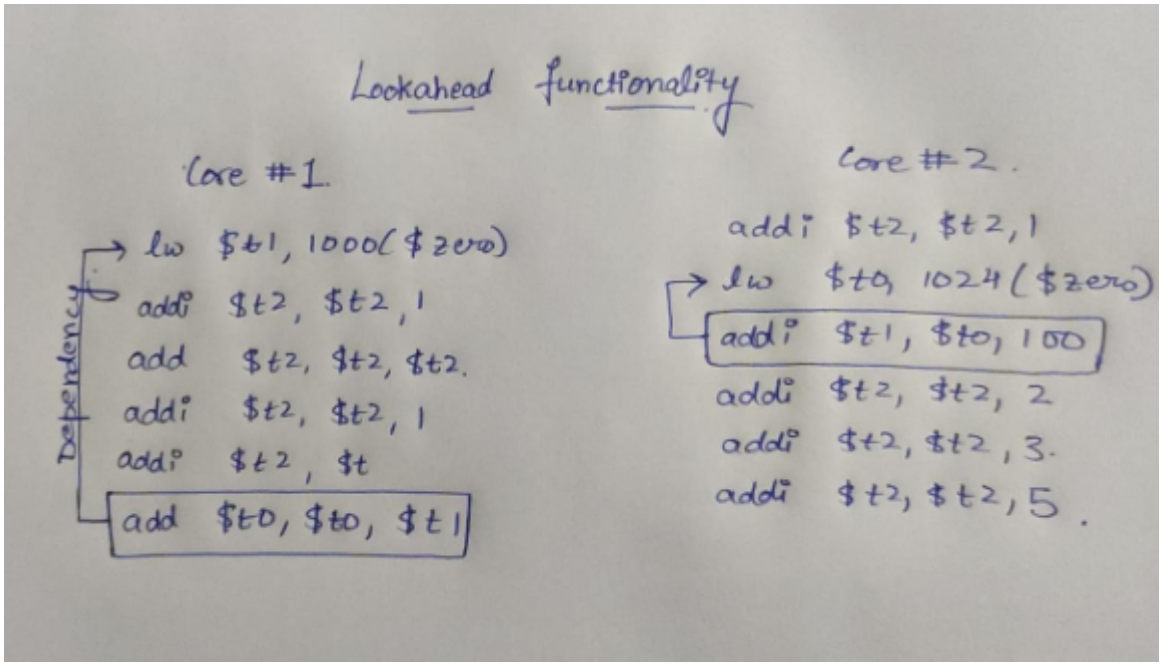
3. Lookahead for increased throughput:

Input:

```
Core #1
lw $t1, 1000($zero)
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
add $t2, $t2, $t2
addi $t2, $t2, 1
addi $t2, $t2, 2
add $t0, $t0, $t1
EOF
```

```
Core #2
addi $t2, $t2, 1
lw $t0, 1024($zero)
addi $t2, $t2, 2
addi $t2, $t2, 2
addi $t2, $t2, 2
addi $t2, $t2, 2
addi $t2, $t2, 2
addi $t2, $t2, 2
addi $t1, $t0, 100
addi $t2, $t2, 2
addi $t2, $t2, 3
addi $t2, $t2, 4
addi $t2, $t2, 5
EOF
```

Working:



Without lookahead:

```
Total Instructions executed in 44 cycles = 24
Average Number of Instructions per cycle (IPC) = 0.545455
Throughput Efficiency 54.5455 %


Program Execution successful
```

With Lookahead:

```
Total Instructions executed in 40 cycles = 24
Average Number of Instructions per cycle (IPC) = 0.6
Throughput Efficiency 60 %


Program Execution successful
```

4. Exception in case of infinite loop / unbounded recurssion:

Consider this case:

```
main:
        addi $t0, $t0, 3
        sw $t0, 1024($zero)
loop:
        beq $t0, $t1, exit
        addi $t1, $t1, 0
        j loop
exit:
EOF
```

Output is an exception:

```
Infinite Loop detected
terminate called after throwing an instance of 'std::exception'
  what():  std::exception
makefile:8: recipe for target 'run' failed
make: *** [run] Aborted (core dumped)
```

5. Exception in a Core should not affect others:

consider core 1 as with invalid input exception:

```
Core #1
lw $t1, 1000($zero)
addi $t2, $t2, 1
addi $t2, $t2, 1
EOF
```

core 2 as:

```
Core #2
addi $t2, $t2, 1
lw $t0, 1024($zero)
addr $t2, $t2, 2
EOF
```

We have exception raised for Core #1:

```
INVALID INSTRUCTION DETECTED!! addr $t2, $t2, 2
Execution of core stopped : 1
```

But the other program executes completely:

```
Total Instructions executed in 37 cycles = 6
Average Number of Instructions per cycle (IPC) = 0.162162
Throughput Efficiency 16.2162 %


Program Execution successful
```

6. Maximizing throughput by parallel processing:

Input of core 1:

```
Core #1
main:
        addi $s0, $zero, 5
        addi $s1, $zero, 0
        addi $s2, $zero, 1
        addi $s3, $zero, 1
        addi $s4, $zero, 1
check:
        slt $t0, $s0, $s2
        beq $t0, $zero, run
        bne $t0, $zero, loopexit
run:
        add $s1, $s1, $s0
        mul $s3, $s3, $s0
        sub $s0, $s0, $s4
        j check
loopexit:
        add $s1, $s1, $zero
        mul $s3, $s3, $s4
exit:
EOF
```

Input of core 2:

```
Core #2
main:
        addi $s0, $zero, 1000
        addi $s1, $zero, 0
        addi $s2, $zero, 10
        addi $t1, $zero, 0
initloop:
        addi $t1, $t1, 1
        addi $s0, $s0, 4
        addi $s1, $s1, 1
        slt $s3, $s1, $s2
        addi $s0, $zero, 1000
        addi $s1, $zero, 0
        addi $s3, $zero, 0
        addi $s2, $zero, 9
sumloop:
        lw $t0, 0($s0)
        addi $s0, $s0, 4
        lw $t1, 0($s0)
        add $t2, $t0, $t1
        addi $s1, $s1, 1
        slt $s3, $s1, $s2
exit:
EOF
```

Input of core 3:

```
Core #3
lw $t1, 1000($zero)
addi $t2, $t2, 1
add $t2, $t2, $t2
addi $t2, $t2, 1
addi $t2, $t2, 2
add $t0, $t0, $t1
EOF
```

Finally

```
Total Instructions executed in 40 cycles = 64
Average Number of Instructions per cycle (IPC) = 1.6
Throughput Efficiency 160 %
```

```
============================================================
Estimated Delay caused due to MRM Operations = 1 clock cycles.
============================================================
```

7.

Input of core 1:

```
Core #1
main:
        addi $s0, $zero, 5
        addi $s1, $zero, 0
        addi $s2, $zero, 1
        addi $s3, $zero, 1
        addi $s4, $zero, 1
check:
        slt $t0, $s0, $s2
        beq $t0, $zero, run
        bne $t0, $zero, loopexit
run:
        add $s1, $s1, $s0
        mul $s3, $s3, $s0
        sub $s0, $s0, $s4
        j check
loopexit:
        add $s1, $s1, $zero
        mul $s3, $s3, $s4
exit:
EOF
```

Input of core 2:

```
Core #2
addi $s0, $s0, 20
sw $s0, 1000($zero)
lw $s1, 1028($zero)
add $s1, $s1, $s0
mul $s2, $s1, $s0
EOF
```

Input of core 3:

```
Core #3
addi $t0, $t0, 15
sw $t0, 1000($zero)
lw $t1, 1000($zero)
add $t1, $t1, $t1
mul $t2, $t1, $t0
EOF
```

Input of core 4:

```
Core #4
main:
        addi $s0, $zero, 1000
        addi $s1, $zero, 2500
        addi $t0, $zero, 1
        addi $t1, $zero, 2
        addi $t2, $zero, 3
        addi $t3, $zero, 4
        sw $t0, 0($s0)  #store 1 at location 1000
        sw $t1, 0($s1)  #store 2 at location 2500
        sw $t2, 4($s0)  #store 3 at location 1004
        sw $t3, 4($s1)  #store 4 at location 2504
        lw $t5, 0($s0)
        lw $t6, 0($s1)
        lw $t7, 4($s0)
        lw $t8, 4($s1)
exit:
EOF
```

Throughput, IPC and total instructions:

```
Total Instructions executed in 114 cycles = 63
Average Number of Instructions per cycle (IPC) = 0.552632
Throughput Efficiency 55.2632 %

Program Execution successful
```

```
================================================================
Estimated Delay caused due to MRM Operations = 41 clock cycles.
================================================================
```

Exceptions:

1. Out of bound access:

```
main:
   sw $s0, 100000000000($s0)
```

```
================================================================
Clock cycle executed: 1
Current Instructiion being executed:    sw $s0, 1000000000000000($s0)
DRAM Request Issued
Segmentation fault
make: *** [makefile:37: run2] Error 139
```

2. Non existent register:

```
main:
    addi $s9, $s12, 12
```

```
INVALID REGISTER DETECTED!! : $s12
Length: 4
```

3. Invalid instruction:

```
main:
    sw5 $s0, 1024
```

```
INVALID INSTRUCTION DETECTED!! sw5
terminate called after throwing an instance of 'std::exception'
```

4. Invalid Branch:

```
main:
  j abc
```

```
Clock cycle executed: 1
Current Instructiion being executed:    j abc

INVALID Branch Detected!! : abc
terminate called after throwing an instance of 'std::exception'
  what():  std::exception
Aborted
```

5. Invalid syntax of instruction:

```
main:
    addi $s0, $s1, $s2
```

```
============================================================
Clock cycle executed: 1
Current Instructiion being executed:    addi $s0, $s1, $s2
terminate called after throwing an instance of 'std::invalid_argument'
  what():  stoi
```