Minor

Anirudha Kulkarni 2019CS50421

Input:

MIPS instruction set with add, addi, sw and lw operations

Approach:

consider

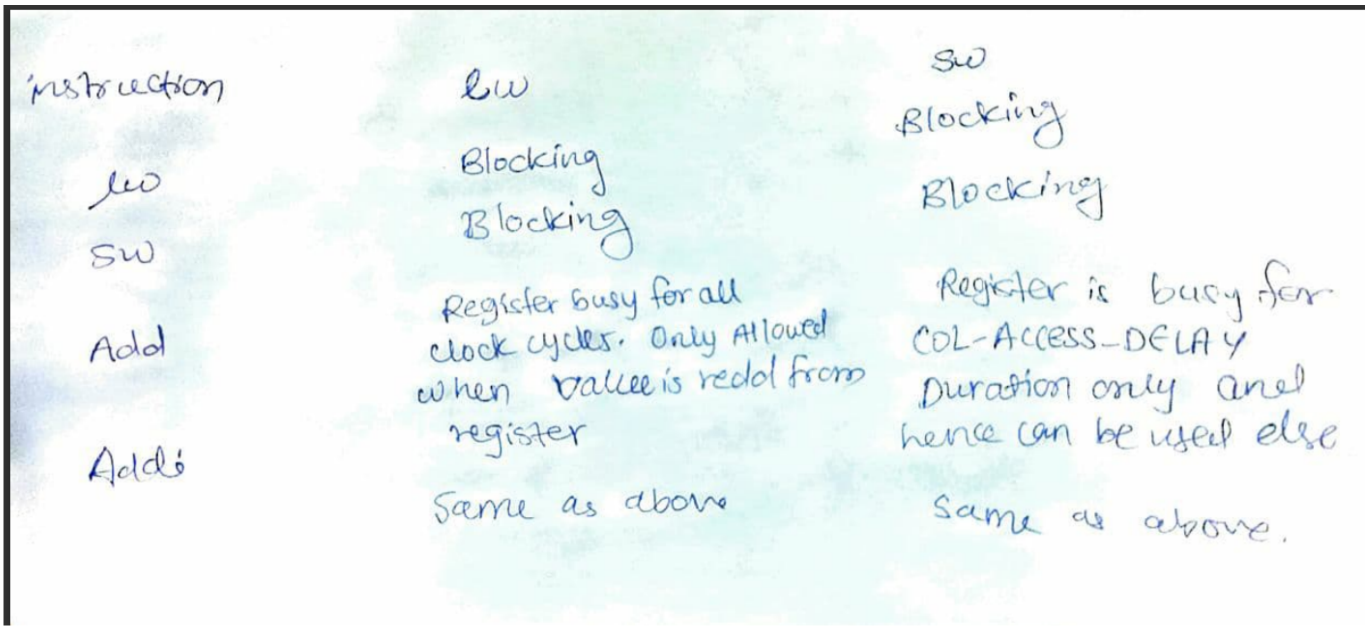
```
lw $t1, 1000
add $t2, $t1, $t2
```

In this case we have `lw` which enguages `$t1` in the `COL_ACCESS_DELAY` duration. In `ROW_ACCESS_DELAY` time we will have `ROWBUFFER` copying elements from `DRAM`. So in this duration we can use this register to perform instructions which involve access to that particular register only or any other register for both read and write operations. But as nothing was mentioned how many cycles are needed for accessing the data from register we assume it will be only during `COL_ACCESS_DELAY` only as mentioned in assignment statement `Copy the data at the column offset from the row buffer to the REGISTER. Time for this operation: COL_ACCESS_DELAY`. Also using the same register in access method does not change the sequential nature of the program. Hence Non-blocking access is possible.

conside

```
lw $t1, 1000
add $t2, $t1, $t2
```

In this case we have loading the address value to the register. Hence we can not allow parallel execution of any instruction involving that register as its value is going to be changed. Hence only instructions with other registers can be safely allowed to execute.



instruction	lw	sw
lw	Blocking	Blocking
sw	Blocking	Blocking
Add	Register busy for all clock cycles. Only Allowed when value is redd from register	Register is busy for COL-ACCESS_DELAY Duration only and hence can be used else
Addi	Same as above	Same as above.

Reasoning For Approach:

1. We need to wait if 2 memory access commands come after each other:

consider

```
lw $t0, 100
lw $t1, 200
```

Here after we start executing 1st line memory unit will be busy for next ROW_ACCESS_DELAY + COL_ACCESS_DELAY or 2*ROW_ACCESS_DELAY + COL_ACCESS_DELAY according to past conditions and hence can not be accessed till then.

PowerPoint Presentation (utah.edu) slide 4

Hence 2nd line will be executed only after complete execution of line 1

2. We can not skip an instruction and execute next instruction

consider,

```
lw $t0, 100
lw $t1, 200
addi $t3, $t3, 10
```

There is scope for optimization by excuting line 3 after line 1 is started to execute. But as it was mentioned in this) piazza post, it can not skip an instruction (in this case line 2) and execute line 3 as it will violet sequential condition of execution

Hence problem reduces to find next instruction with registers only with no conflict and execute it in parallel

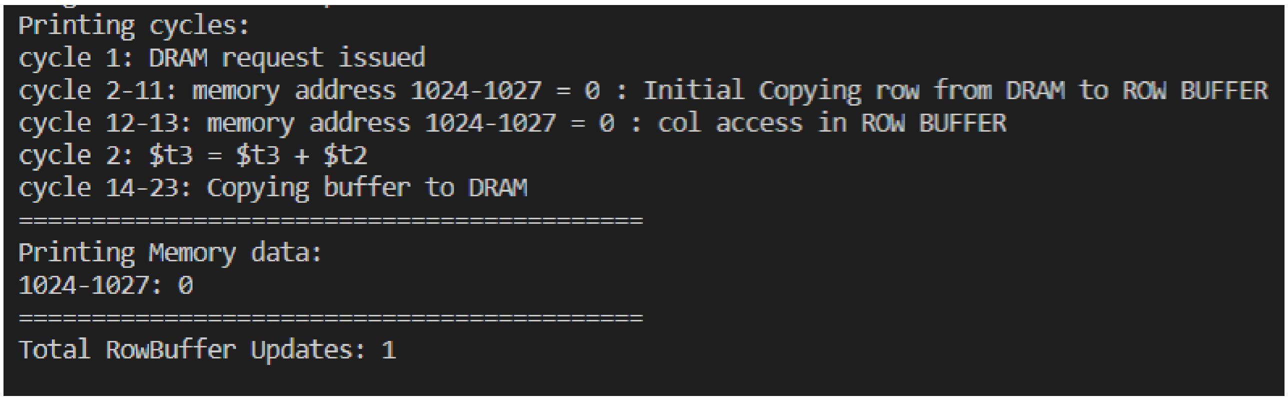
3. Copying back buffer:

Row stored in the buffer needs to be copied back to the DRAM after last execution is over

Testing:

1. Parallel execution possible

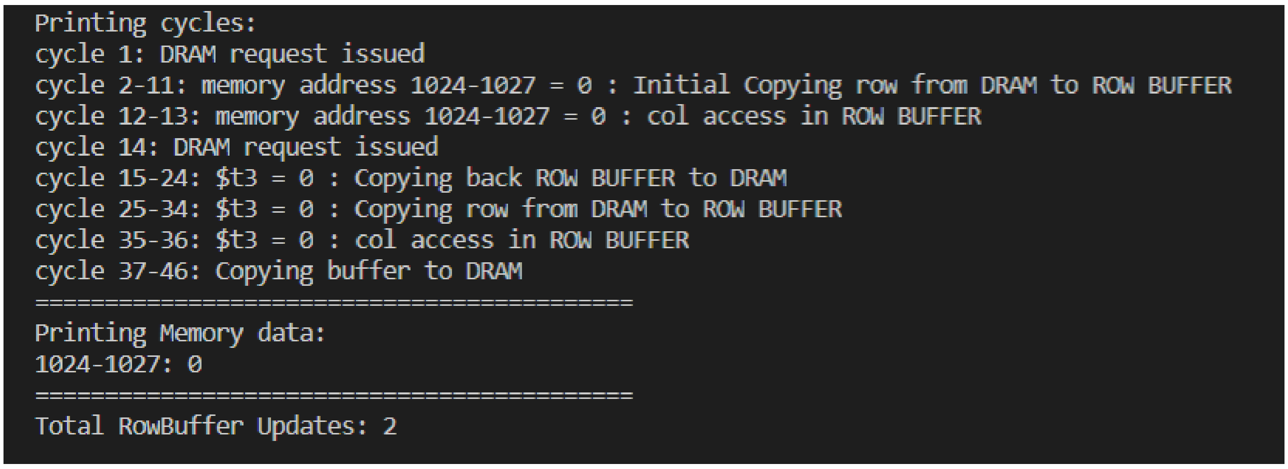
```
sw $t1, 1024
add $t3, $t3, $t2
```



```
Printing cycles:
cycle 1: DRAM request issued
cycle 2-11: memory address 1024-1027 = 0 : Initial Copying row from DRAM to ROW BUFFER
cycle 12-13: memory address 1024-1027 = 0 : col access in ROW BUFFER
cycle 2: $t3 = $t3 + $t2
cycle 14-23: Copying buffer to DRAM
===========================================
Printing Memory data:
1024-1027: 0
===========================================
Total RowBuffer Updates: 1
```

2. simlutaneous memory instructions

```
sw $t1, 1024
lw $t3, 9999
```

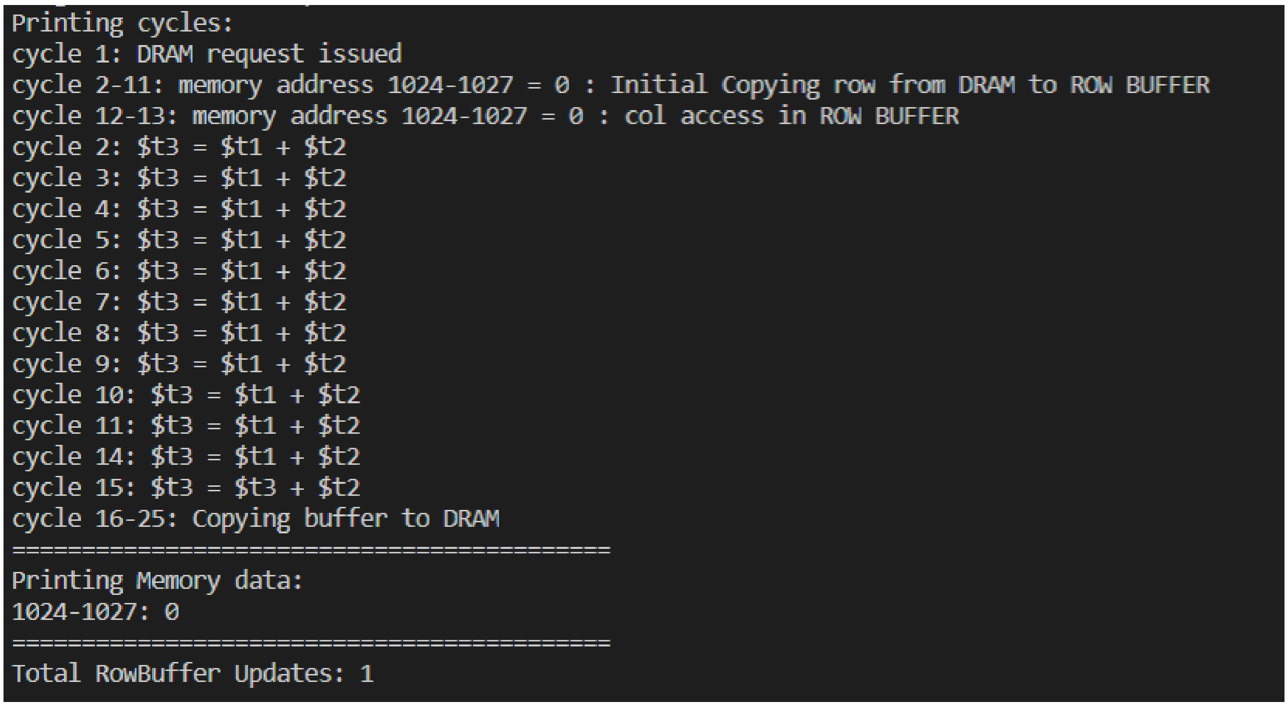


```
Printing cycles:
cycle 1: DRAM request issued
cycle 2-11: memory address 1024-1027 = 0 : Initial Copying row from DRAM to ROW BUFFER
cycle 12-13: memory address 1024-1027 = 0 : col access in ROW BUFFER
cycle 14: DRAM request issued
cycle 15-24: $t3 = 0 : Copying back ROW BUFFER to DRAM
cycle 25-34: $t3 = 0 : Copying row from DRAM to ROW BUFFER
cycle 35-36: $t3 = 0 : col access in ROW BUFFER
cycle 37-46: Copying buffer to DRAM
===========================================
Printing Memory data:
1024-1027: 0
===========================================
Total RowBuffer Updates: 2
```

3. jump at 13th instruction

```
sw $t1, 1024
add $t3, $t1, $t2
add $t3, $t1, $t2
add $t3, $t1, $t2
add $t3, $t1, $t2
```

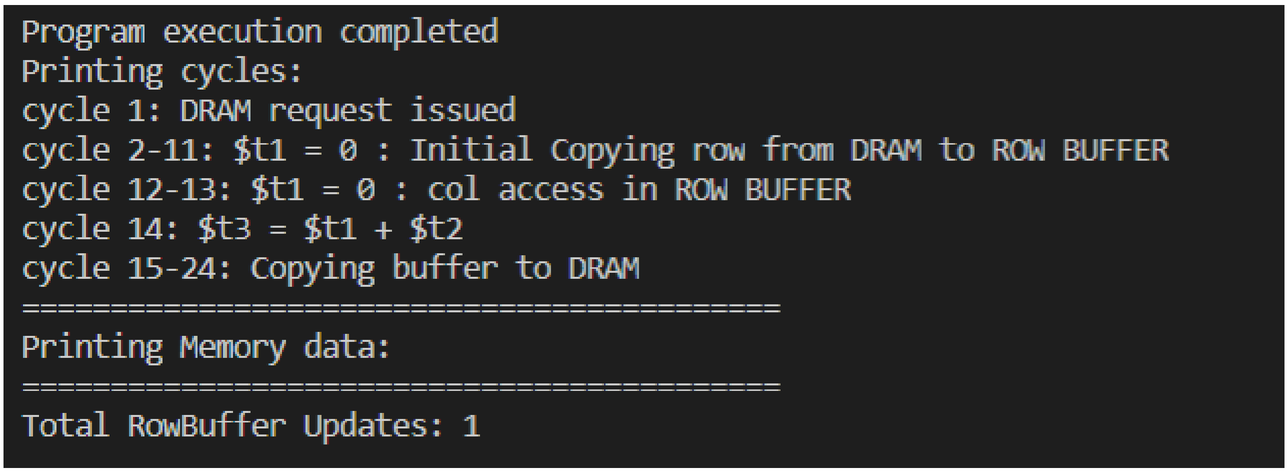
```
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t1, $t2
  add $t3, $t3, $t2
```



```
Printing cycles:
cycle 1: DRAM request issued
cycle 2-11: memory address 1024-1027 = 0 : Initial Copying row from DRAM to ROW BUFFER
cycle 12-13: memory address 1024-1027 = 0 : col access in ROW BUFFER
cycle 2: $t3 = $t1 + $t2
cycle 3: $t3 = $t1 + $t2
cycle 4: $t3 = $t1 + $t2
cycle 5: $t3 = $t1 + $t2
cycle 6: $t3 = $t1 + $t2
cycle 7: $t3 = $t1 + $t2
cycle 8: $t3 = $t1 + $t2
cycle 9: $t3 = $t1 + $t2
cycle 10: $t3 = $t1 + $t2
cycle 11: $t3 = $t1 + $t2
cycle 14: $t3 = $t1 + $t2
cycle 15: $t3 = $t3 + $t2
cycle 16-25: Copying buffer to DRAM
=============================================
Printing Memory data:
1024-1027: 0
=============================================
Total RowBuffer Updates: 1
```

4. nsaf

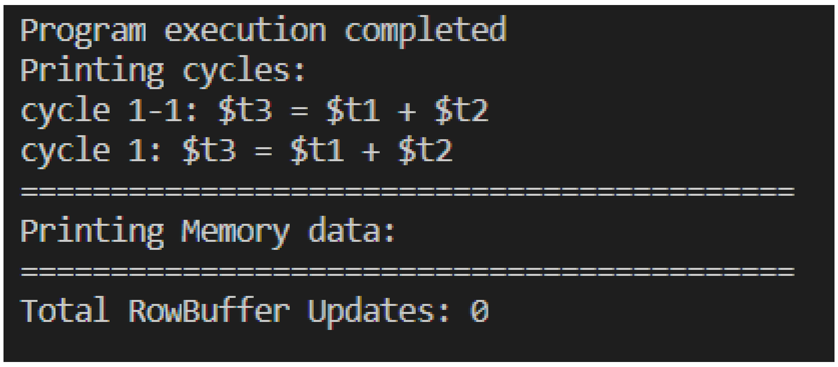
```
  lw $t1, 1024
  add $t3, $t1, $t2
  add $t3, $t1, $t2
```



```
Program execution completed
Printing cycles:
cycle 1: DRAM request issued
cycle 2-11: $t1 = 0 : Initial Copying row from DRAM to ROW BUFFER
cycle 12-13: $t1 = 0 : col access in ROW BUFFER
cycle 14: $t3 = $t1 + $t2
cycle 15-24: Copying buffer to DRAM
=============================================
Printing Memory data:
=============================================
Total RowBuffer Updates: 1
```

5. No sw/lw command:

```
add $t3, $t1, $t2
```



```
Program execution completed
Printing cycles:
cycle 1-1: $t3 = $t1 + $t2
cycle 1: $t3 = $t1 + $t2
===========================================
Printing Memory data:
===========================================
Total RowBuffer Updates: 0
```

6.