Minor

Anirudha Kulkarni 2019CS50421

Input:

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

Build Instructions:

Program can be run easily with make instructions:

1. To Build: `make build`
2. To run executable from build `make run`
3. To remove build file `make clean`

All these steps can be performed by `make all` which will remove previous build, create new one and give output.

Custom `ROW_ACCESS_DELAY` and `COL_ACCESS_DELAY` can be provided by navigating to `build` directory and

```
./main ../input.txt 100 45
```

which will execute the program with `ROW_ACCESS_DELAY` as 100 and `COL_ACCESS_DELAY` as 45 Provide input in input.txt or give it as command line argument. Defaults used when `make all` is used are:

```
`ROW_ACCESS_DELAY`: 10
`COL_ACCESS_DELAY`: 2
input location : 'input.txt'
```

Unoptimised code can be run by:

```
make unopt
```

Approach:

consider

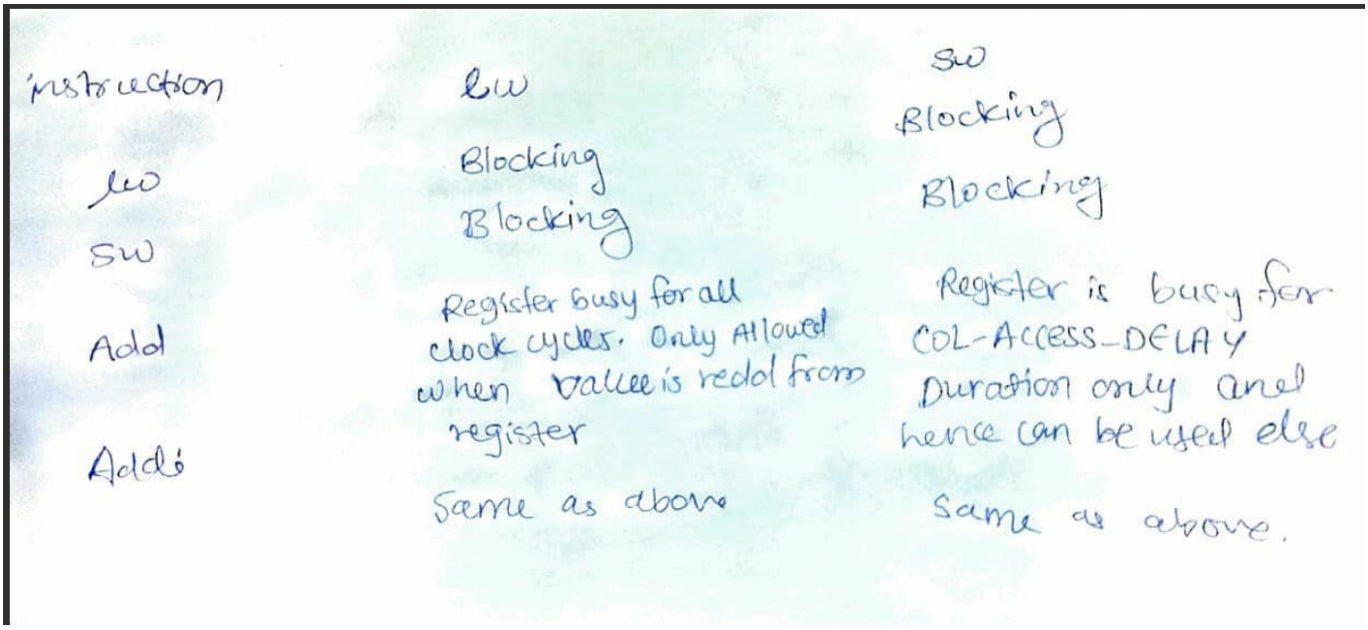
```
lw $t1, 1000
add $t2, $t1, $t2
```

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.



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

Assumptions:

1. Registers are accessed only during `COL_ACCESS_DELAY` duration
2. `ROW_ACCESS_DELAY` > `COL_ACCESS_DELAY` and both are non zero
3. Instructions dont consume cycle to be accessed from memory
4. Maximum instructions and data values are 2**19 so that overall memory will not exceed 2 **20

Strengths:

1. Maintains the sequential nature of the program rather than manipulating it via lookaheads.
2. Reduced complexity compared with priority queue which requires checking in distinct future values which can invlove lookaheadd of `O(N)` plus backtracking of `O(N)` and hence increase evaluation time in case of larger inputs. Ex: `10 sec` vs `30 sec` for program with `10**10` lines which is order of a general MIPS processor handles.
3. Modular and easily understandable approach due to Object oriented approach
4. Debugging friendly approach which makes easier to follow the execution of program as lookahead is `O(ROW_ACCESS_DELAY)`

Weakness:

1. In contrast to approach with queue for prioritising request this will not be able to optimise the cycles in the distinct future as notion of lookahead and backtracking was avoided. This can be severe in case of specific operations like `lw $t1, 1000` and high values of `ROW_ACCESS_DELAY`. In such cases we will waste lot of cycles which could have been saved in the future. But such instructions are very specific in occurence.

2. Object oriented model might take large space even for smaller inputs. All the variables are needed to be initialised and take constant space in contrast to functional approach which creates variables as we need it but looses modularity and order of computations in a MIPS processor is much larger than such case.

Testing:

Strategy:

- We shall start with all cases with the branching explained in the approach. All the standard inputs will be verified.
- We shall then consider corner cases
- Then we shall compare the optimisation with unoptimised code so as to verify the optimisation of cycles.

Cases:

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 due to bllocking by reading of register. Also consumes 25 cycles even though large number of instructions are present due to Non-blocking access to next instructions.

```
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. lw blocks access

```
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
```

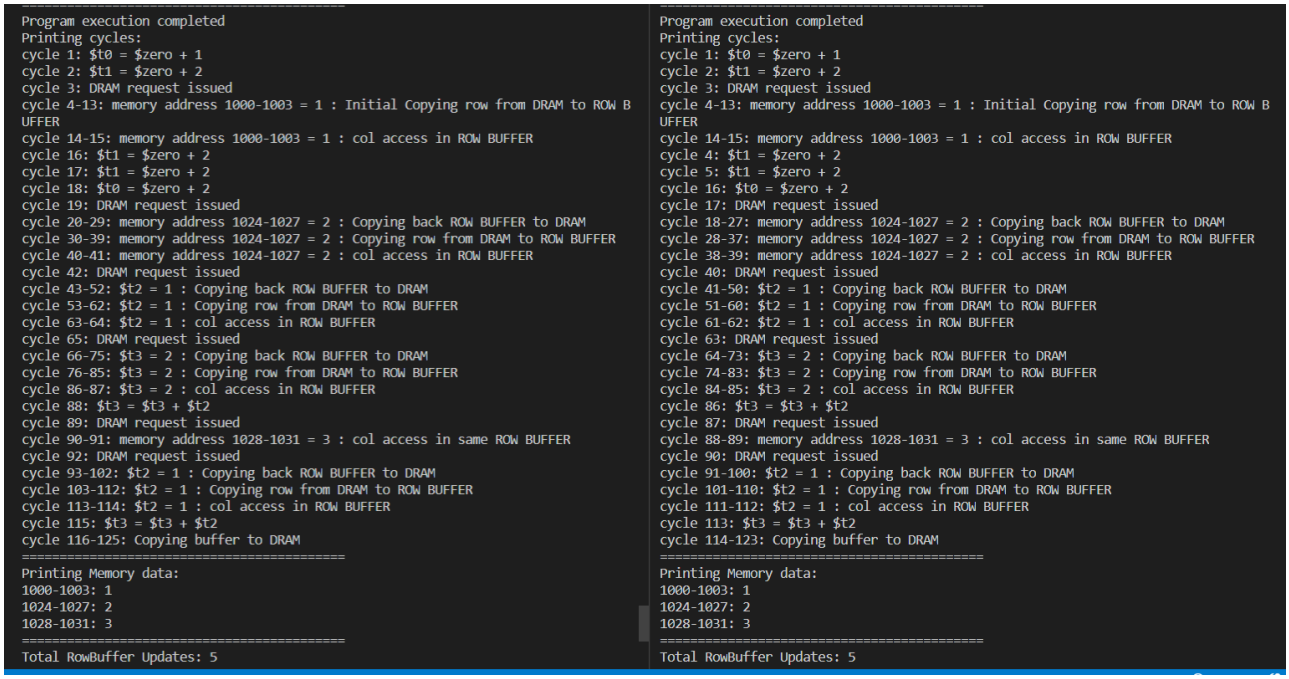
5. No sw/lw command:

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

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

6. Unoptimised vs optimised

```
main:
addi $t0, $zero, 1
addi $t1, $zero, 2
sw $t0, 1000
addi $t1, $zero, 2
addi $t1, $zero, 2
addi $t0, $zero, 2
sw $t1, 1024
lw $t2, 1000
lw $t3, 1024
add $t3, $t3, $t2
sw $t3, 1028
lw $t2, 1000
add $t3, $t3, $t2
exit:
```



```
Program execution completed
Printing cycles:
cycle 1: $t0 = $zero + 1
cycle 2: $t1 = $zero + 2
cycle 3: DRAM request issued
cycle 4-13: memory address 1000-1003 = 1 : Initial Copying row from DRAM to ROW B
UFFER
cycle 14-15: memory address 1000-1003 = 1 : col access in ROW BUFFER
cycle 16: $t1 = $zero + 2
cycle 17: $t1 = $zero + 2
cycle 18: $t0 = $zero + 2
cycle 19: DRAM request issued
cycle 20-29: memory address 1024-1027 = 2 : Copying back ROW BUFFER to DRAM
cycle 30-39: memory address 1024-1027 = 2 : Copying row from DRAM to ROW BUFFER
cycle 40-41: memory address 1024-1027 = 2 : col access in ROW BUFFER
cycle 42: DRAM request issued
cycle 43-52: $t2 = 1 : Copying back ROW BUFFER to DRAM
cycle 53-62: $t2 = 1 : Copying row from DRAM to ROW BUFFER
cycle 63-64: $t2 = 1 : col access in ROW BUFFER
cycle 65: DRAM request issued
cycle 66-75: $t3 = 2 : Copying back ROW BUFFER to DRAM
cycle 76-85: $t3 = 2 : Copying row from DRAM to ROW BUFFER
cycle 86-87: $t3 = 2 : col access in ROW BUFFER
cycle 88: $t3 = $t3 + $t2
cycle 89: DRAM request issued
cycle 90-91: memory address 1028-1031 = 3 : col access in same ROW BUFFER
cycle 92: DRAM request issued
cycle 93-102: $t2 = 1 : Copying back ROW BUFFER to DRAM
cycle 103-112: $t2 = 1 : Copying row from DRAM to ROW BUFFER
cycle 113-114: $t2 = 1 : col access in ROW BUFFER
cycle 115: $t3 = $t3 + $t2
cycle 116-125: Copying buffer to DRAM
========================================
Printing Memory data:
1000-1003: 1
1024-1027: 2
1028-1031: 3
========================================
Total RowBuffer Updates: 5
```

```
Program execution completed
Printing cycles:
cycle 1: $t0 = $zero + 1
cycle 2: $t1 = $zero + 2
cycle 3: DRAM request issued
cycle 4-13: memory address 1000-1003 = 1 : Initial Copying row from DRAM to ROW B
UFFER
cycle 14-15: memory address 1000-1003 = 1 : col access in ROW BUFFER
cycle 4: $t1 = $zero + 2
cycle 5: $t1 = $zero + 2
cycle 16: $t0 = $zero + 2
cycle 17: DRAM request issued
cycle 18-27: memory address 1024-1027 = 2 : Copying back ROW BUFFER to DRAM
cycle 28-37: memory address 1024-1027 = 2 : Copying row from DRAM to ROW BUFFER
cycle 38-39: memory address 1024-1027 = 2 : col access in ROW BUFFER
cycle 40: DRAM request issued
cycle 41-50: $t2 = 1 : Copying back ROW BUFFER to DRAM
cycle 51-60: $t2 = 1 : Copying row from DRAM to ROW BUFFER
cycle 61-62: $t2 = 1 : col access in ROW BUFFER
cycle 63: DRAM request issued
cycle 64-73: $t3 = 2 : Copying back ROW BUFFER to DRAM
cycle 74-83: $t3 = 2 : Copying row from DRAM to ROW BUFFER
cycle 84-85: $t3 = 2 : col access in ROW BUFFER
cycle 86: $t3 = $t3 + $t2
cycle 87: DRAM request issued
cycle 88-89: memory address 1028-1031 = 3 : col access in same ROW BUFFER
cycle 90: DRAM request issued
cycle 91-100: $t2 = 1 : Copying back ROW BUFFER to DRAM
cycle 101-110: $t2 = 1 : Copying row from DRAM to ROW BUFFER
cycle 111-112: $t2 = 1 : col access in ROW BUFFER
cycle 113: $t3 = $t3 + $t2
cycle 114-123: Copying buffer to DRAM
========================================
Printing Memory data:
1000-1003: 1
1024-1027: 2
1028-1031: 3
========================================
Total RowBuffer Updates: 5
```

We see that instruction 4 and 5 are executed in cycle 4,5 vs 16,17 in unoptimised code.