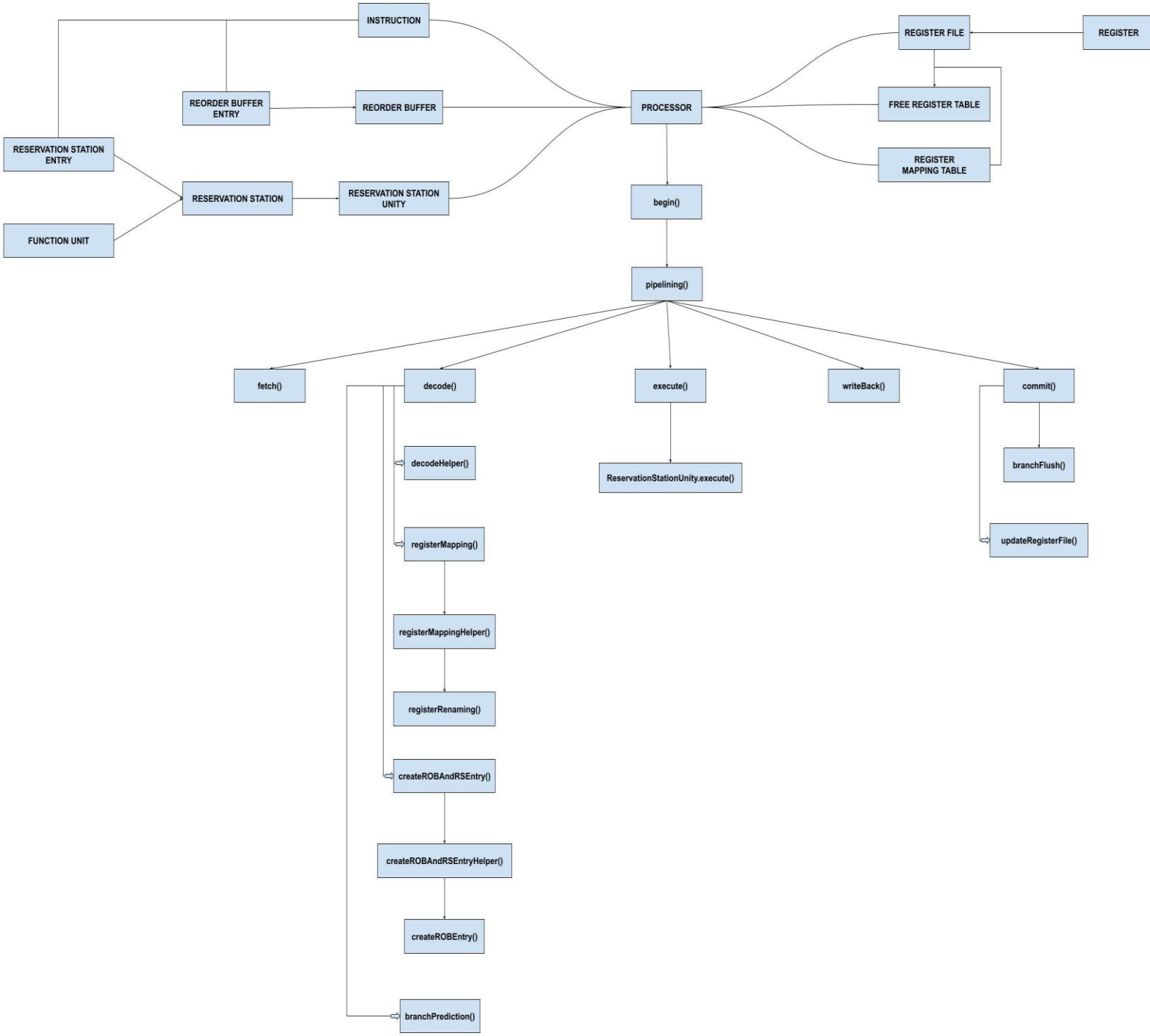
CS 2410 Computer Architecture
Spring 2022
Course Project

Project Module Design



Every entry with Capital Letters in the flow chart is a class thus can have instances in the attached Class.

Every lower-case entry is a function.

Not all functions of each Class have been shown in the diagram due to the complexity and sheer number of functions that exists within them. Below is the explanation of how each of these components are implemented.

The program starts from main.py file which creates an instance of Processor Class and calls begin method from Processor Class to start the simulation.

CONSTRUCTOR: The constructor method, creates instances of classes Reservation Station Unity that has all reservation station buffers and functional units mapped to it, Reorder Buffer class with empty entries, Register File that has register instances instantiated inside its constructor, Free Register Table, Register Mapping Table. It also includes all the necessary variables and files instantiation like instruction file, memory file, decode queue, branch predictor bit, etc.

BEGIN: The begin method calls the pipelining method in a while loop until pipelining is finished i.e., when no entries in ROB and no more instructions are being fetched or decoded.

PIPELINING: The pipelining method executes each stage of Pipeline being fetch, decode, execute, writeback and commit. It also keeps track of condition for pipeline being finished.

FETCH: The fetch method initiates with default program counter and starts fetching instructions and putting them in a decode queue.

DECODE:

1. All the instructions in decode queue are first parsed according to their instruction type and then mapped to an instance of Instruction Class.
2. Within the decoding stage, as no original mapping was provided, a mapping with physical registers with architected registers is made. For every destination a new register is used to ensure no false dependencies are left.
3. In the case of BNE instruction, the branch prediction mechanism is called that checks BTB for the entry, if none exists it creates one. This mechanism returns the new value for program counter.
4. After the conventional decode step is done, a separate method is called from within the decode stage to instantiate the ROB and RS entry.
5. First, ROB is checked for availability, if not all instructions are stalled till a entry is freed. Second, Reservation Station availability is checked followed by same rules. When both are available, an entry to ROB is made that is attached to Reservation Station entry’s destination.

EXECUTE: The Reservation Station Unity class includes an execution method that is launched through this stage. This method has ripple effect as it caused each Reservation Station to execute its functional unit. An entry is executed if it has all the necessary register values required to run the instruction. If no such entry exists, the process is stalled for that particular entry only as Out of order execution is implemented.

WRITEBACK: All the instruction stages are being updated in ROB Table, when the state of an entry turns from execute to execute complete. The data from that entry is entered in Common Data Bus for all Reservation Station Listening.

COMMIT: In the final stage, only the ROB head is checked if it is in Writeback stage, if it is, it is committed, freeing all the renamed registers and updating the values inside the register table. The store instructions make an entry to memory in this stage to ensure no un-committed store instruction make irreversible changes.

In case of wrong branch prediction, the branch predictor bit is switched and all the instructions are flushed.

Reservation Station Unity Class Execution Explanation:

The reservation station unity class has all the buffers and functional units attached to it as an instance of Reservation Station Class.

The Reservation Station Unity execute method launches the execute method of each functional unit.

If the Reservation Station has an entry that is ready to run and the functional unit is free. It is executed otherwise it waits till all entries are available.

Analysis

Demo Code

```
Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4'], 'F4': ['p6'], '$0': ['p10']}

Free Registers: ['p21', 'p22', 'p30', 'p29', 'p3', 'p28', 'p27', 'p26', 'p25',
'p24', 'p31', 'p5', 'p7', 'p8', 'p9', 'p11', 'p12', 'p13', 'p14', 'p15',
'p16', 'p17', 'p18', 'p19', 'p20', 'p23']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 0, None], 'p3': [False, 12, None], 'p4': [False, 0, None], 'p5':
[False, 168.0, ROB4], 'p6': [False, 27, None], 'p7': [False, 27.0, ROB6],
'p8': [False, 0, ROB8], 'p9': [False, 100, ROB9], 'p10': [False, 0, None],
'p11': [False, 111, ROB11], 'p12': [False, 60.0, ROB12], 'p13': [False, 3,
ROB13], 'p14': [False, 63.0, ROB14], 'p15': [False, -8, ROB0], 'p16': [False,
108, ROB1], 'p17': [False, 14, ROB3], 'p18': [False, 0, ROB4], 'p19': [False,
27, ROB5], 'p20': [False, 0, ROB6], 'p21': [False, 0, ROB8], 'p22': [False, 0,
```

```
ROB9], 'p23': [False, 0, ROB11], 'p24': [False, 0, ROB12], 'p25': [False, 0,
ROB13], 'p26': [False, 0, ROB14], 'p27': [False, 0, ROB0], 'p28': [False, 0,
ROB1], 'p29': [False, 0, ROB3], 'p30': [False, 0, ROB4], 'p31': [False, 0,
None], }

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 0, 'F2': 12, 'F0': 0,
'F4': 27, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]', '[88 :
0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]', '[136 :
0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 : 0]', '[184 :
0]', '[192 : 0]', '[200 : 12]']
```

Total Cycles: 34

Stalls due to {'RS': 4, 'ROB': 9, 'CDB': 0}

Study the effect of changing the issue and commit width to 2. That is setting NW=NB=2 rather than 4.

```
[31]: STATE: FETCH

Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4'], 'F4': ['p6'], '$0': ['p10']}

Free Registers: ['p22', 'p5', 'p31', 'p30', 'p29', 'p3', 'p28', 'p27',
'p26', 'p25', 'p24', 'p7', 'p8', 'p9', 'p11', 'p12', 'p13', 'p14', 'p15',
'p16', 'p17', 'p18', 'p19', 'p20', 'p21', 'p23']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 0, None], 'p3': [False, 12, None], 'p4': [False, 0, None], 'p5':
[False, 168.0, ROB6], 'p6': [False, 0, None], 'p7': [False, 27.0, ROB6],
'p8': [False, 0, ROB8], 'p9': [False, 100, ROB9], 'p10': [False, 0, None],
'p11': [False, 111, ROB11], 'p12': [False, 0.0, ROB12], 'p13': [False, 111,
ROB13], 'p14': [False, 63.0, ROB14], 'p15': [False, -8, ROB0], 'p16':
[False, 108, ROB1], 'p17': [False, 14, ROB3], 'p18': [False, 0, ROB4],
'p19': [False, 27, ROB5], 'p20': [False, 0, ROB6], 'p21': [False, 0, ROB8],
'p22': [False, 0, ROB9], 'p23': [False, 0, ROB11], 'p24': [False, 0,
ROB12], 'p25': [False, 0, ROB13], 'p26': [False, 0, ROB14], 'p27': [False,
0, ROB0], 'p28': [False, 0, ROB1], 'p29': [False, 0, ROB3], 'p30': [False,
0, ROB4], 'p31': [False, 0, ROB5], }

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 0, 'F2': 12, 'F0': 0,
'F4': 27, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]',
'[88 : 0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]',
```

```
'[136 : 0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 :
0]', '[184 : 0]', '[192 : 0]', '[200 : 12]']
```

Toal Cycles: 40

Stalls due to {'RS': 0, 'ROB': 10, 'CDB': 3}

As less instructions were being allowed in the common data bus. It took more cycles as the instructions were being taking longer to commit. And it can also be observed that in this case there are stalls introduced due to CDB being full.

Study the effect of changing the fetch/decode width. That is setting NF = 2 rather than 4

```
Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4'], 'F4': ['p6'], '$0': ['p10']}

Free Registers: ['p15', 'p16', 'p17', 'p18', 'p19', 'p20', 'p21', 'p22', 'p7',
'p5', 'p31', 'p30', 'p29', 'p3', 'p28', 'p27', 'p26', 'p25', 'p24', 'p8',
'p9', 'p11', 'p12', 'p13', 'p14', 'p23']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 0, None], 'p3': [False, 12, None], 'p4': [False, 0, None], 'p5':
[False, 168.0, ROB6], 'p6': [False, 0, None], 'p7': [False, 27.0, ROB8], 'p8':
[False, 0, ROB8], 'p9': [False, 100, ROB9], 'p10': [False, 0, None], 'p11':
[False, 111, ROB11], 'p12': [False, 0.0, ROB12], 'p13': [False, 111, ROB13],
'p14': [False, 63.0, ROB14], 'p15': [False, -8, ROB0], 'p16': [False, 108,
ROB1], 'p17': [False, 14, ROB3], 'p18': [False, 0, ROB4], 'p19': [False, 27,
ROB5], 'p20': [False, 0, ROB6], 'p21': [False, 0, ROB8], 'p22': [False, 0,
ROB9], 'p23': [False, 0, ROB11], 'p24': [False, 0, ROB12], 'p25': [False, 0,
ROB13], 'p26': [False, 0, ROB14], 'p27': [False, 0, ROB0], 'p28': [False, 0,
ROB1], 'p29': [False, 0, ROB3], 'p30': [False, 0, ROB4], 'p31': [False, 0,
ROB5], }

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 0, 'F2': 12, 'F0': 0,
'F4': 0, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]', '[88 :
0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]', '[136 :
0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 : 0]', '[184 :
0]', '[192 : 0]', '[200 : 12]']
```

Toal Cycles: 38

Stalls due to {'RS': 3, 'ROB': 8, 'CDB': 0}

Here the cycles increased can be dedicated to the fact, only few instructions could be fetched, thus limiting the number of instructions that can process in out of order execution.

Study the effect of changing the number of reorder buffer entries. That is setting NR = 4, 8, and 32

NR = 4

```
Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4'], 'F4': ['p6'], '$0': ['p10']}

Free Registers: ['p11', 'p12', 'p13', 'p14', 'p15', 'p16', 'p17', 'p18',
'p19', 'p20', 'p21', 'p22', 'p23', 'p24', 'p25', 'p26', 'p27', 'p28',
'p29', 'p30', 'p31', 'p5', 'p7', 'p8', 'p9']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 116, None], 'p3': [False, 12, None], 'p4': [False, 24.0, None],
'p5': [False, 16, ROB0], 'p6': [False, 8, None], 'p7': [False, 24.0, ROB2],
'p8': [False, 0, ROB0], 'p9': [False, 116, ROB1], 'p10': [False, 0, None],
'p11': [False, 0, None], 'p12': [False, 0, None], 'p13': [False, 0, None],
'p14': [False, 0, None], 'p15': [False, 0, None], 'p16': [False, 0, None],
'p17': [False, 0, None], 'p18': [False, 0, None], 'p19': [False, 0, None],
'p20': [False, 0, None], 'p21': [False, 0, None], 'p22': [False, 0, None],
'p23': [False, 0, None], 'p24': [False, 0, None], 'p25': [False, 0, None],
'p26': [False, 0, None], 'p27': [False, 0, None], 'p28': [False, 0, None],
'p29': [False, 0, None], 'p30': [False, 0, None], 'p31': [False, 0, None],
}

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 116, 'F2': 12, 'F0':
24.0, 'F4': 8, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]',
'[88 : 0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]',
'[136 : 0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 :
0]', '[184 : 0]', '[192 : 0]', '[200 : 12]']
```

Toal Cycles: 19

Stalls due to {'RS': 0, 'ROB': 16, 'CDB': 0}

Due to very low size of ROB, the simulation failed to execute properly. After once, entering instructions in the ROB, these instructions did not commit for several cycles thus resulting in overall halt of the program as no more instruction were left to fetch till bne could be evaluated. Thus after remaining idle for several cycles the simulation broke.

NR = 8

```
Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4', 'p17'], 'F4': ['p6'], '$0': ['p10']}
```

```
Free Registers: ['p14', 'p15', 'p16', 'p19', 'p18', 'p20', 'p21', 'p22',
'p23', 'p24', 'p25', 'p26', 'p27', 'p28', 'p29', 'p30', 'p31', 'p5', 'p7',
'p8', 'p9', 'p11', 'p12', 'p13']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 0, None], 'p3': [False, 12, None], 'p4': [False, 0, None], 'p5':
[False, 60.0, ROB4], 'p6': [False, 3, None], 'p7': [False, 3.0, ROB6], 'p8':
[False, 8, ROB0], 'p9': [False, 108, ROB1], 'p10': [False, 0, None], 'p11':
[False, 5, ROB3], 'p12': [False, 0, ROB4], 'p13': [False, 3, ROB5], 'p14':
[False, 0, ROB6], 'p15': [False, 0, ROB0], 'p16': [False, 0, ROB1], 'p17':
[True, 0, ROB3], 'p18': [False, 0, ROB4], 'p19': [False, 0, ROB5], 'p20':
[False, 0, None], 'p21': [False, 0, None], 'p22': [False, 0, None], 'p23':
[False, 0, None], 'p24': [False, 0, None], 'p25': [False, 0, None], 'p26':
[False, 0, None], 'p27': [False, 0, None], 'p28': [False, 0, None], 'p29':
[False, 0, None], 'p30': [False, 0, None], 'p31': [False, 0, None], }

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 0, 'F2': 12, 'F0': 0,
'F4': 3, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]', '[88 :
0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]', '[136 :
0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 : 0]', '[184 :
0]', '[192 : 0]', '[200 : 12]']
```

Toal Cycles: 30

Stalls due to {'RS': 0, 'ROB': 16, 'CDB': 0}

In comparison to RS, the overhead was ROB, as it was the main cause of all the halts in the program. Still, the program was able to run quickly as less entries in ROB, less time was spent in executing wrong branch predicted instructions.

NR = 32

```
Register Mapping: {'R0': ['p0'], 'R1': ['p1'], 'R2': ['p2'], 'F2': ['p3'],
'F0': ['p4'], 'F4': ['p6'], '$0': ['p10']}

Free Registers: ['p21', 'p22', 'p5', 'p31', 'p30', 'p29', 'p3', 'p28', 'p27',
'p26', 'p25', 'p24', 'p7', 'p8', 'p9', 'p11', 'p12', 'p13', 'p14', 'p15',
'p16', 'p17', 'p18', 'p19', 'p20', 'p23']

Register Values: {'p0': [False, 0, None], 'p1': [False, 0, None], 'p2':
[False, 100, None], 'p3': [False, 12, None], 'p4': [False, 111, None], 'p5':
[False, 120.0, ROB6], 'p6': [False, 27, None], 'p7': [False, 128.0, ROB6],
'p8': [False, 16, ROB8], 'p9': [False, 116, ROB9], 'p10': [False, 0, None],
'p11': [False, 5, ROB11], 'p12': [False, 60.0, ROB12], 'p13': [False, 3,
ROB13], 'p14': [False, 63.0, ROB14], 'p15': [False, 8, ROB16], 'p16': [False,
```

```
108, ROB17], 'p17': [False, 14, ROB19], 'p18': [False, 168.0, ROB20], 'p19':
[False, 27, ROB21], 'p20': [False, 195.0, ROB22], 'p21': [False, 0, ROB24],
'p22': [False, 100, ROB25], 'p23': [False, 111, ROB27], 'p24': [False, 0,
ROB28], 'p25': [False, 2, ROB29], 'p26': [False, 0, ROB30], 'p27': [False, -8,
ROB0], 'p28': [False, 0, ROB1], 'p29': [False, 0, ROB3], 'p30': [False, 0,
ROB4], 'p31': [False, 0, ROB5], }

Architected Register Values: {'R0': 0, 'R1': 0, 'R2': 100, 'F2': 12, 'F0':
111, 'F4': 27, '$0': 0}

Main Memory: ['[0 : 111]', '[8 : 14]', '[16 : 5]', '[24 : 10]', '[32 : 0]',
'[40 : 0]', '[48 : 0]', '[56 : 0]', '[64 : 0]', '[72 : 0]', '[80 : 0]', '[88 :
0]', '[96 : 0]', '[104 : 0]', '[112 : 0]', '[120 : 0]', '[128 : 0]', '[136 :
0]', '[144 : 0]', '[152 : 0]', '[160 : 0]', '[168 : 0]', '[176 : 0]', '[184 :
0]', '[192 : 0]', '[200 : 12]']
```

Toal Cycles: 34

Stalls due to {'RS': 12, 'ROB': 0, 'CDB': 0}

Here all the overhead came due to RS entries, as ROB has 32 entries allowing as many as possible entries.

Overall, it can be observed that increasing the size of ROB does not necessarily increase efficiency as most entries would be idle if the reservation station is the cause of stalls.

Decrease the entries in ROB is helpful as well, because of Out of order execution, wrong branch predicted instructions can take precedence over the legit instructions thus decreasing the size stops this problem from happening.

Overall, the default conditions were good and ROB with size 8 had best efficiency in terms of cycles needed to execute.