**Key data structures used in the RISC-V Simulator code:**

1. **Memory (Dictionary):**

The memory dictionary stores memory contents, with memory addresses as keys and memory values as values. This data structure allows the simulator to read and write to memory locations during program execution.

Example:

memory = {

0: 111,

8: 14,

16: 5,

24: 10,

100: 2,

# Additional memory entries...

}

In this example, memory addresses 0, 8, 16, 24, and 100 are mapped to their respective values 111, 14, 5, 10, and 2.

1. **Registers (Dictionary):**

The registers dictionary represents CPU registers, where register names (e.g., R0, R1) are used as keys and register values are stored as values. This data structure enables the simulator to read from and write to CPU registers during program execution.

Example:

python

Copy code

registers = {

'R0': 0,

'R1': 24,

'R2': 124,

# Additional register entries...

}

In this example, registers R0, R1, and R2 are initialized with the values 0, 24, and 124, respectively.

1. **Labels (Dictionary):**

The labels dictionary stores labels encountered in the input assembly code along with their corresponding line numbers. This data structure facilitates the handling of branch instructions by providing quick access to the target line numbers.

Example:

python

Copy code

labels = {

'loop': 7,

# Additional label entries...

}

In this example, the label loop is associated with line number 7 in the input assembly code.

1. **Instructions (List):**

The instructions list holds parsed instructions from the input assembly file. Each instruction is stored as a string in the list, allowing the simulator to iterate over the instructions during program execution.

Example:

python

Copy code

instructions = [

'addi R1, R0, 24',

'addi R2, R0, 124',

'fld F2, 200(R0)',

# Additional instruction entries...

]

In this example, the first three instructions parsed from the input assembly code are stored in the instructions list.

**b. The results and analysis of Comparative analysis above**

**1) Study the effect of changing the issue and commit width to 2. That is setting**

NW=NB=2 rather than 4

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

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

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

**3) Study the effect of changing the NI to 4 instead of 16.**

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

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

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

**B- For 8**

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

**C – For 32**

Register Values:

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 0

R8: 0

R9: 0

R10: 0

R11: 0

R12: 0

R13: 0

R14: 0

R15: 0

R16: 0

R17: 0

R18: 0

R19: 0

R20: 0

R21: 0

R22: 0

R23: 0

R24: 0

R25: 0

R26: 0

R27: 0

R28: 0

R29: 0

R30: 0

R31: 0

Memory Contents:

0: 2

8: 14

16: 5

24: 10

100: 2

108: 27

116: 3

124: 8

200: 12

2: 2000

4: 2

Program Counter (PC): 0

1. **Register Values and Memory Contents:**

The register values remain unchanged, indicating that no instructions have modified them.

Memory contents also remain consistent across different runs, suggesting that memory operations are not affecting the data stored in memory.

1. **Program Counter (PC):**

The program counter (PC) remains at 0 for all runs, indicating that the processor is starting execution from the beginning of the program.

1. **Observations:**

The processor is executing the same program multiple times with different parameter configurations.

There are no observable changes in register values, memory contents, or PC across runs, suggesting that the program execution behavior remains consistent.

1. **Analysis:**

Since the register values, memory contents, and PC remain constant, it's challenging to perform a detailed analysis of the impact of parameter variations solely based on these sample results.

Additional performance metrics such as IPC, throughput, and resource utilization is required to assess the effects of parameter changes more comprehensively.

Further testing with a variety of benchmark programs and detailed performance monitoring would be necessary to draw meaningful conclusions about the optimal parameter configurations for the processor design.

1. **Recommendations:**

Additional benchmarking with a diverse set of workloads to capture a broader range of performance characteristics to be done.

Implement performance monitoring tools within the simulator to collect detailed metrics during execution.