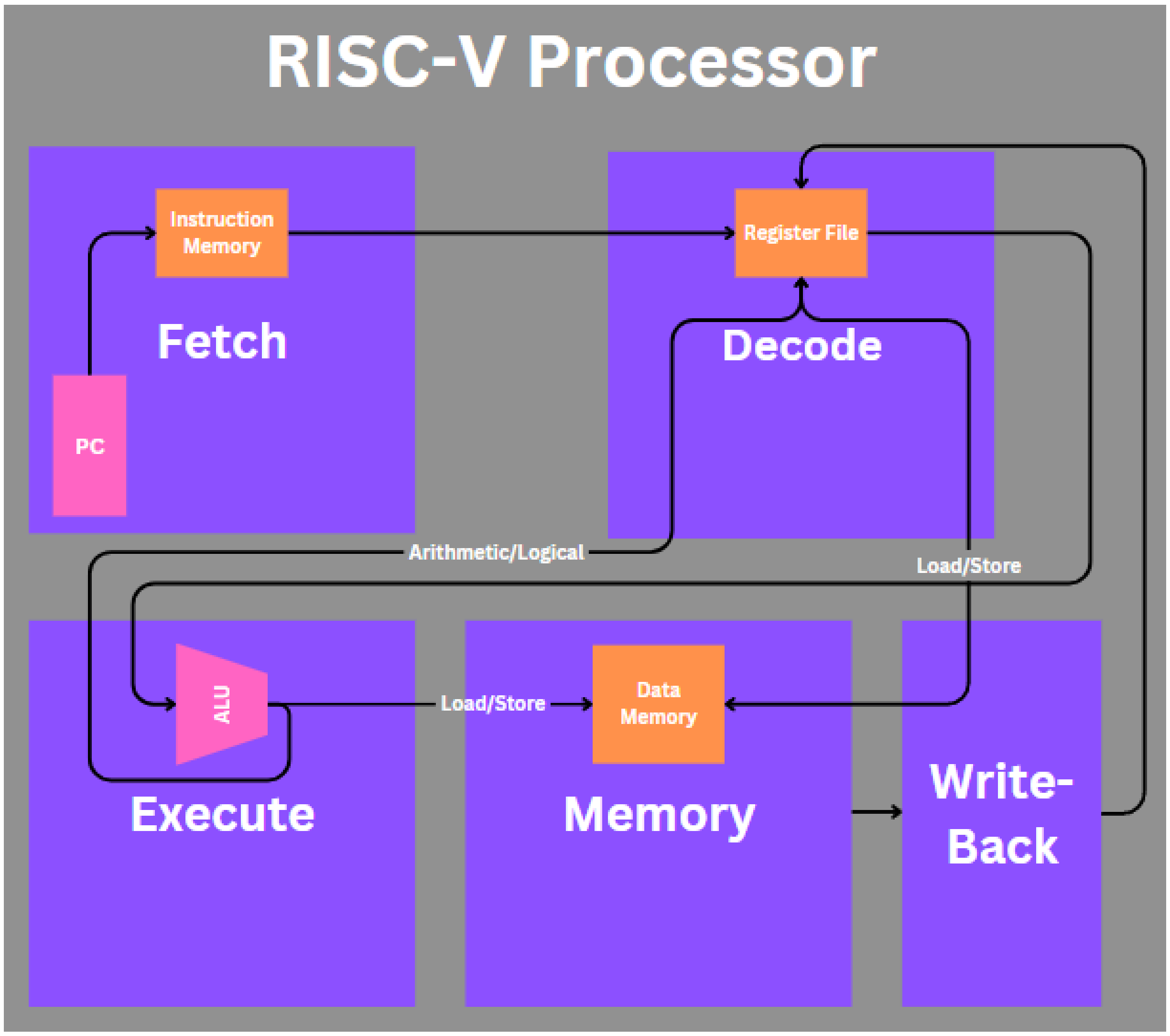
Jack Gao Francisco Soriano Raul Graterol Medina

SCHOOL RISC-V Verification

By Team: Shift-Register

Brief Description with High Level Diagram:



Instruction Fetch (IF)

All of the instructions of the RISC-V Single Cycle Processor start by using the **program counter (PC)** to supply the instruction's address to the **instruction memory** which fetches the corresponding instruction and then through to the processor sequence.

Instruction Decode/Instruction Fetch (ID)

After the instruction is fetched, the opcode field (which defines the type of instruction - arithmetic, load, store, branch) and the register operands fields (which specify which registers are needed for the operation) used by the instruction are decoded. Then, using these fields and specifications, the processor now knows which registers to **read** from the register file and retrieves the operands (data/values) stored in those registers.

Execution
These operands can then be operated on to compute a memory address for a (load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction (ALU)), or an equality check (for a branch).

With regards to **Memory Address Calculation,** for load/store instructions the ALU calculates the address in memory where data should be read or written to. For **Arithmetic/Logical Operation** instructions (i.e. addition, subtraction, etc.) the ALU performs the calculation. Regarding **Branch Evaluation**, for branch instructions, the ALU checks for equality or other conditions between 2 register operands to determine if a jump to a new instruction address is needed.

Memory Access
Memory Access occurs only if applicable. If the instruction is arithmetic or logical, the ALU result (such as a sum or logical AND) must be written to a **destination register** in the register file. However, there is different behavior when regarding Load/Store instructions.

For a **load** instruction, the ALU output is treated as a **memory address** where the data to be loaded is located at. After this output is produced, the processor accesses memory at that address and loads the data from memory into the specified register in the register file. Load instructions modify the register file.

For a **store** instruction, the ALU output is also used as a **memory address**, where the data will be stored. The processor then writes data from a specified register into this calculated memory address. Store instructions move data from a register to memory.

Write-Back
The ALU result is written back into the register file so that it can be used for subsequent instructions if said result derives from an arithmetic or logical instruction. Additionally, data from memory in load instructions is also written back into the register file.

PC Update
Regarding Branch instructions, the ALU calculates the next PC based on the branch condition. More specifically, if the Branch condition is met, the new PC address is calculated by adding the branch offset to the current PC or from an adder that increments the current PC by 4 bytes.

Note
Given that this is a single-cycle implementation of a RISC-V Processor, each instruction is passed through each stage of the processor sequentially within a single-clock cycle.

Design Description:

The design is a minimalist System-on-Chip that brings together an educational CPU, called School-RISC V, which is based on the RISC-V architecture, and a Read-Only Memory. In this design, the ROM is pre-loaded with a program that calculates Fibonacci numbers, which the CPU executes step by step. Essentially, the CPU fetches instructions from the ROM, decodes them, and then carries out the operations required to compute the Fibonacci sequence.

The School-RISC V CPU itself is a simplified RISC V processor designed specifically for educational purposes. It is a processor that handles essential tasks, including arithmetic, branching, and memory access with some basic optimizations. This design makes it ideal for understanding fundamental CPU operations and focusing on the basics of how a CPU executes instructions.

The ALU module is a core component of the schoolRISCV CPU, handling basic arithmetic and logical tasks essential for the processor's function. It operates on two input values and performs different operations depending on the control signals it receives, providing a result along with a flag indicating if the result is zero. Supporting fundamental functions like addition, bitwise operations, shifts, comparisons, and subtraction, this ALU continuously evaluates inputs and updates its output accordingly. Designed with simplicity, this ALU fulfills the primary needs of an educational RISC-V CPU, making it suitable for foundational learning and straightforward processor designs.

Test Plan:

For our test plan, we plan to test and verify the individual components rather than the whole CPU and the function of calculating the fibonacci numbers. More specifically, we decided to run the testbench verifications on the modules for ALU, decoder, register file, and control. We decided to leave out the CPU, SOC, and register with rst as we prioritized the individual functionalities of the modules rather than the integrated modules. The register with rst was left since it was a D flip flop with a reset signal. We concentrated on the main functionalities of the individual modules through various ways of verification such as coverage, constraints and pass/fail cases within the testbench codes.

When it comes to constraints, most of the testbenches that were written have some constraints that were more specifically geared towards the limitations of the opcode pools as listed in the header files. From the header files given within this project, the opcodes may be 4 bits or 7 bits depending on the design and while it may be those bit widths, the opcodes pool does not expand all 16 combinations for 4 bits or all 128 possible combinations for 7 bits. Keeping this in mind, we mostly have our constraints to limit the opcodes to those listed within the header files of this project. For the randomization parts, we let the randomizer choose whatever as we would simply verify the output results from the randomized generated inputs. For the special case of the control testbench, we made some distribution weights on the constraints for the function3 and function7 variables since it has some variations according to the header files. While there is a set of specific ones from the header file, there can also be cases where they can be random of any combinations, hence we set the weighted distribution to be half random of any and half contains the listed defined variables.

For the number of tests, we let the randomized runs be however many it takes to obtain the full coverage of what we wanted to cover. On top of obtaining the full functional coverage that we desire, we made some expected test cases inside the testbenches where we drive specific inputs into the module and check the expected output from the module to ensure its functionality and verify that the module works as intended. During the testing of the inputs, we also implemented the pass/fail logic with the expected and the actual outputs to verify that the module. Overall, our test plan mainly consists of testing the individual modules of the RISC-V processor to ensure that they work as intended where we constrained the inputs and generated randomized cases to obtain most of our desired coverage.

Coverage Summary:

For our coverage summary, we have increased the vector sizes in order to cover %100 of our coverages. For the main part, within our testbenches, we are trying to obtain the coverage summary for the valid opcodes that were listed within the header files of the project. For the register file, we wanted to perform the coverage for all of the address locations of the registers as there were 32 possible register locations due to the address width being 5 bits wide. For the less important variables, we left the coverpoint to remain a wildcard since they were not as important to cover. For example, the ALU testbench were focused on covering all the valid opcodes into the ALU module whereas the inputs A and B were left as wildcards as those can simply be randomly generated and left as they are as they are simply inputs to the module and would be used to generate the output by the ALU module by whatever the opcode input is. For the decoder, we left it be any instructions as all the instructions are worth covering over while we let the constraint fulfill the opcode pool for the instruction generation.

Pass/Fail Summary:

For the pass/fail case of our verification, we attempted to make all the input cases pass to ensure the functionality of the design module. We have attached various inputs and outputs to a task to ensure the actual results with the expected results. For the purpose of debugging and verifying, we would display both the inputs and outputs to show that the functionality was correct while fixing any issues when the test case failed from the results. For different modules, our pass/fail would vary as for the decoder, we would manually dissect the instruction bits and match the expected decoded bits with the actual output bits coming from the module instantiation. For the pass/fail cases that we manually input, we had the testbench test for some edge cases such as the zero cases for the ALU testbench since rather than letting the randomization achieve such results, we manually input numbers that would result in the edge case and see how the module would handle such case. Overall, we have made it through all the testbenches with all the pass/fail cases to be passed as we wanted to ensure that the input to output cases were passed to verify that the design functions as intended.

Written Summary:

Overall, this project was a great experience for allowing us to demonstrate verification topics on a project. When it comes to learning, we had little experience with the RISC-V processor so performing verification on this project allowed us to explore more of the RISC-V processor and learn the RISC-V architecture and its instruction encoding. In terms of learning verification, we applied various methods of verification onto modules of the whole RISC-V processor to ensure their functionality and this process gave us solid experience on the fundamentals of applying verification on designs, much similar to how verification engineers perform such tasks in the industry. On top of the experience with the RISC-V processor in HDL, this project allowed us to be creative with however we wanted to approach the verification of the design, thus allowing us to create the testbenches however we would like to verify the designs on top of learning how individual modules function inside the processor architecture that we choose to work with.

Link to EDAPlayground:

https://www.edaplayground.com/x/c95L

Appendix:

ALU:
Coverage Data:

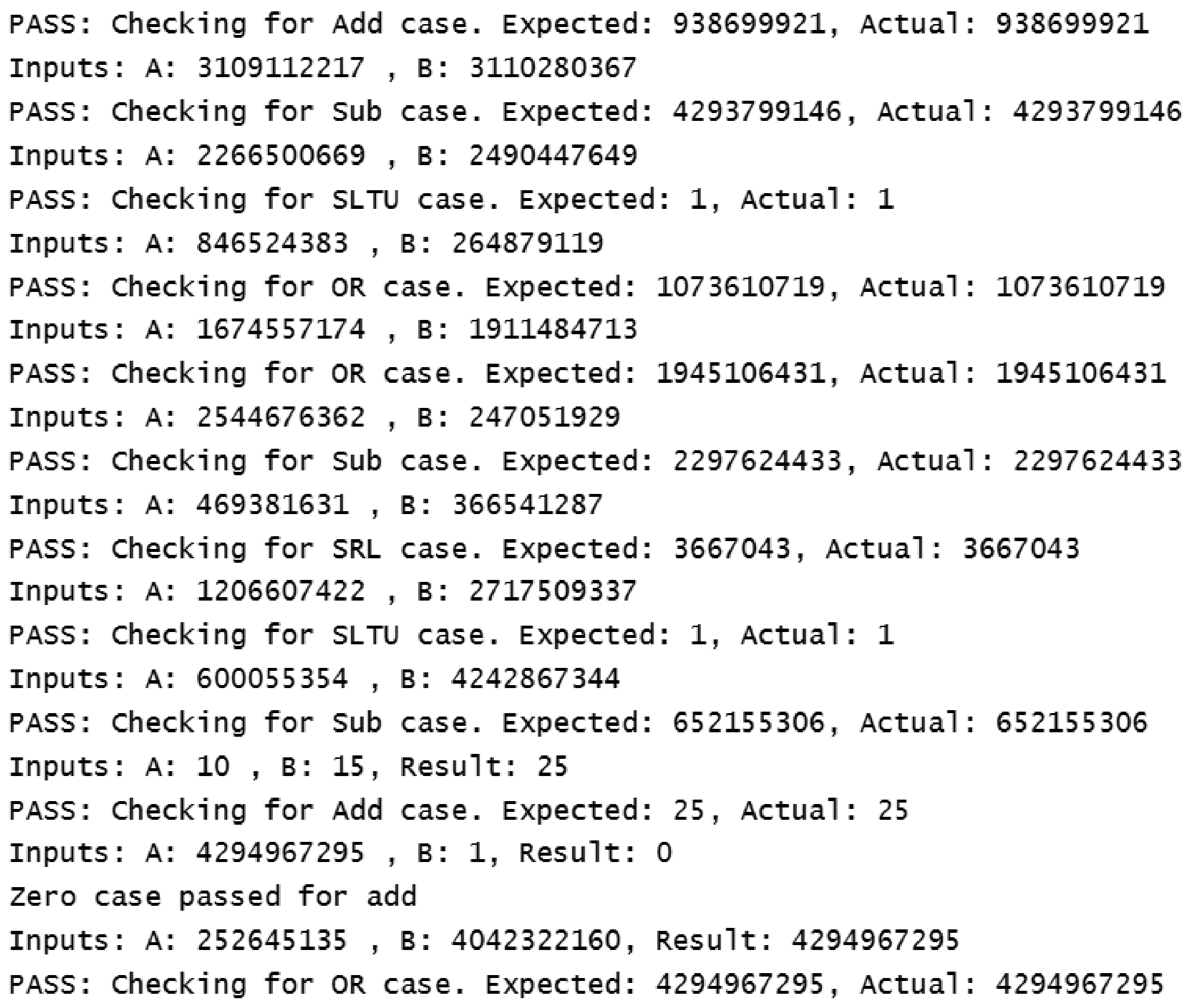
```
-------------------------------------------------------------------------------

Summary for Group   tb::cg_cross



CATEGORY  EXPECTED UNCOVERED COVERED PERCENT
Variables 7        0         7       100.00
Crosses   5        0         5       100.00


Variables for Group  tb::cg_cross


VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST AUTO BIN MAX COMMENT
cp_oper  5        0         5       100.00  100  1      1        0
cp_srcA  1        0         1       100.00  100  1      1        0
cp_srcB  1        0         1       100.00  100  1      1        0


Crosses for Group  tb::cg_cross


CROSS       EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST PRINT MISSING COMMENT
cg_cross_cc 5        0         5       100.00  100  1      1        0


-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

Summary for Variable cp_oper


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 5        0         5       100.00


User Defined Bins for cp_oper


Bins

NAME COUNT AT LEAST
sub  10    1
SLTU 9     1
SRL  9     1
OR   13    1
add  9     1


-------------------------------------------------------------------------------
```

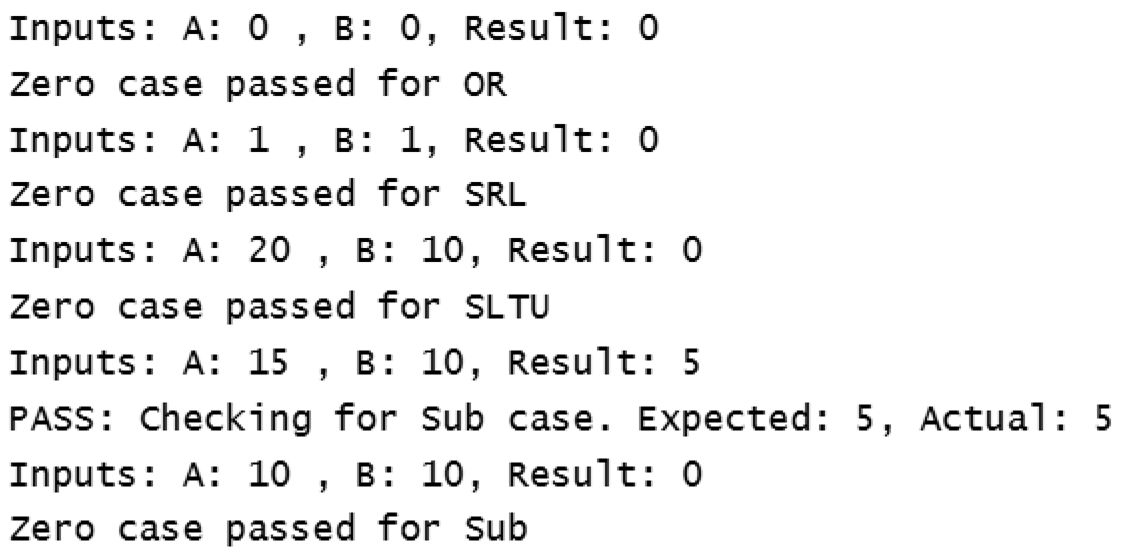
```
--------------------------------------------------------------------------------

Summary for Variable cp_srcA


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 1        0         1       100.00


User Defined Bins for cp_srcA


Bins

NAME COUNT AT LEAST
A    50    1


--------------------------------------------------------------------------------

Summary for Variable cp_srcB


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 1        0         1       100.00


User Defined Bins for cp_srcB


Bins

NAME COUNT AT LEAST
B    50    1


--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

Summary for Cross cg_cross_cc


Samples crossed: cp_oper cp_srcA cp_srcB
CATEGORY                           EXPECTED UNCOVERED COVERED PERCENT MISSING
Automatically Generated Cross Bins 5        0         5       100.00


Automatically Generated Cross Bins for cg_cross_cc


Bins

cp_oper cp_srcA cp_srcB COUNT AT LEAST
sub     A       B       10    1
SLTU    A       B       9     1
SRL     A       B       9     1
OR      A       B       13    1
add     A       B       9     1


Tests

Total Coverage Summary
SCORE  GROUP
100.00 100.00


Total tests in report: 1
-------------------------------------------------------------------------------
```

Pass/Fail Data:

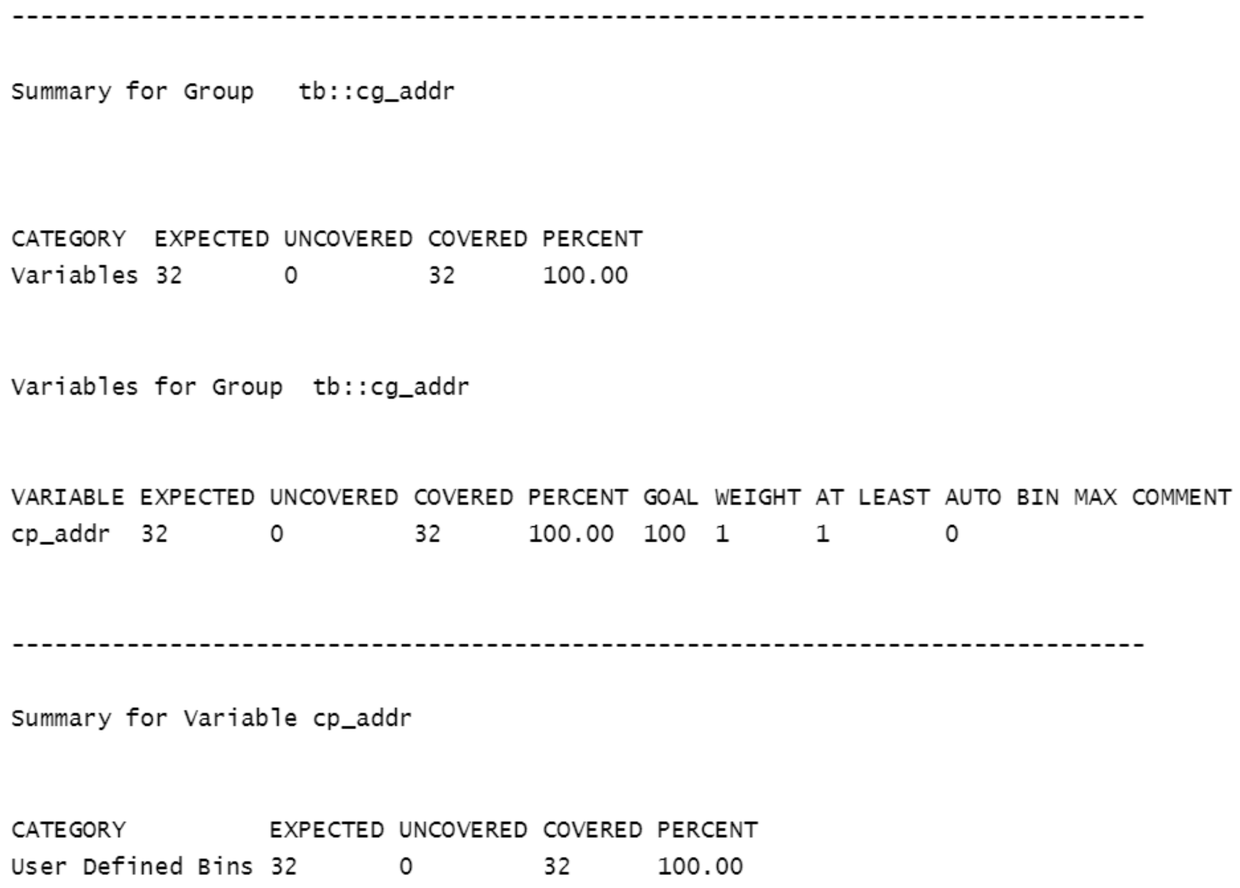


```
PASS: Checking for Add case. Expected: 938699921, Actual: 938699921
Inputs: A: 3109112217 , B: 3110280367
PASS: Checking for Sub case. Expected: 4293799146, Actual: 4293799146
Inputs: A: 2266500669 , B: 2490447649
PASS: Checking for SLTU case. Expected: 1, Actual: 1
Inputs: A: 846524383 , B: 264879119
PASS: Checking for OR case. Expected: 1073610719, Actual: 1073610719
Inputs: A: 1674557174 , B: 1911484713
PASS: Checking for OR case. Expected: 1945106431, Actual: 1945106431
Inputs: A: 2544676362 , B: 247051929
PASS: Checking for Sub case. Expected: 2297624433, Actual: 2297624433
Inputs: A: 469381631 , B: 366541287
PASS: Checking for SRL case. Expected: 3667043, Actual: 3667043
Inputs: A: 1206607422 , B: 2717509337
PASS: Checking for SLTU case. Expected: 1, Actual: 1
Inputs: A: 600055354 , B: 4242867344
PASS: Checking for Sub case. Expected: 652155306, Actual: 652155306
Inputs: A: 10 , B: 15, Result: 25
PASS: Checking for Add case. Expected: 25, Actual: 25
Inputs: A: 4294967295 , B: 1, Result: 0
Zero case passed for add
Inputs: A: 252645135 , B: 4042322160, Result: 4294967295
PASS: Checking for OR case. Expected: 4294967295, Actual: 4294967295
```



```
Inputs: A: 0 , B: 0, Result: 0
Zero case passed for OR
Inputs: A: 1 , B: 1, Result: 0
Zero case passed for SRL
Inputs: A: 20 , B: 10, Result: 0
Zero case passed for SLTU
Inputs: A: 15 , B: 10, Result: 5
PASS: Checking for Sub case. Expected: 5, Actual: 5
Inputs: A: 10 , B: 10, Result: 0
Zero case passed for Sub
```

Register File:
Coverage Data:



```
--------------------------------------------------------------------------------

Summary for Group   tb::cg_addr



CATEGORY  EXPECTED UNCOVERED COVERED PERCENT
Variables 32       0         32      100.00


Variables for Group  tb::cg_addr


VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST AUTO BIN MAX COMMENT
cp_addr  32       0         32      100.00  100  1      1        0


--------------------------------------------------------------------------------

Summary for Variable cp_addr


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 32       0         32      100.00
User Defined Bins for cp_addr


Bins

NAME          COUNT AT LEAST
all_addrs_00 8      1
all_addrs_01 16     1
all_addrs_02 2      1
all_addrs_03 14     1
all_addrs_04 6      1
all_addrs_05 4      1
all_addrs_06 4      1
all_addrs_07 10     1
all_addrs_08 12     1
all_addrs_09 12     1
all_addrs_0a 2      1
all_addrs_0b 2      1
all_addrs_0c 6      1
all_addrs_0d 12     1
all_addrs_0e 6      1
all_addrs_0f 8      1
all_addrs_10 10     1
all_addrs_11 2      1
all_addrs_12 8      1
all_addrs_13 4      1
all_addrs_14 14     1
all_addrs_15 2      1
all_addrs_16 8      1
all_addrs_17 10     1
all_addrs_18 10     1
all_addrs_19 10     1
all_addrs_1a 12     1
all_addrs_1b 6      1
```

```
all_addrs_1c 2      1
all_addrs_1d 6      1
all_addrs_1e 14     1
all_addrs_1f 14     1

Group : tb::cg_cross

================================================================================
Group : tb::cg_cross
================================================================================
SCORE  WEIGHT GOAL   AT LEAST AUTO BIN MAX PRINT MISSING
100.00 1      100    1        64           64


Source File(s) :

/home/runner/testbench.sv



--------------------------------------------------------------------------------

Summary for Group   tb::cg_cross



CATEGORY  EXPECTED UNCOVERED COVERED PERCENT
Variables 33       0         33      100.00
Crosses   32       0         32      100.00


Variables for Group  tb::cg_cross


VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST AUTO BIN MAX COMMENT
cp_addr  32       0         32      100.00  100  1      1        0
cp_data  1        0         1       100.00  100  1      1        0
```

```
Crosses for Group  tb::cg_cross


CROSS       EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST PRINT MISSING COMMENT
cg_cross_cc 32       0         32      100.00  100  1      1        0


-------------------------------------------------------------------------------

Summary for Variable cp_addr


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 32       0         32      100.00


User Defined Bins for cp_addr
```

```
Bins

NAME         COUNT AT LEAST
all_addrs_00 4     1
all_addrs_01 8     1
all_addrs_02 1     1
all_addrs_03 7     1
all_addrs_04 3     1
all_addrs_05 2     1
all_addrs_06 2     1
all_addrs_07 5     1
all_addrs_08 6     1
all_addrs_09 6     1
all_addrs_0a 1     1
all_addrs_0b 1     1
all_addrs_0c 3     1
all_addrs_0d 6     1
all_addrs_0e 3     1
all_addrs_0f 4     1
all_addrs_10 5     1
all_addrs_11 1     1
all_addrs_12 4     1
all_addrs_13 2     1
all_addrs_14 7     1
all_addrs_15 1     1
all_addrs_16 4     1
all_addrs_17 5     1
all_addrs_18 5     1
all_addrs_19 5     1
all_addrs_1a 6     1
all_addrs_1b 3     1
all_addrs_1c 1     1
all_addrs_1d 3     1
all_addrs_1e 7     1
all_addrs_1f 7     1
```

```
-------------------------------------------------------------------------------

Summary for Variable cp_data


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 1        0         1       100.00


User Defined Bins for cp_data


Bins

NAME     COUNT AT LEAST
all_data 128   1


-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

Summary for Cross cg_cross_cc


Samples crossed: cp_addr cp_data
CATEGORY                           EXPECTED UNCOVERED COVERED PERCENT MISSING
Automatically Generated Cross Bins 32       0         32      100.00


Automatically Generated Cross Bins for cg_cross_cc
```

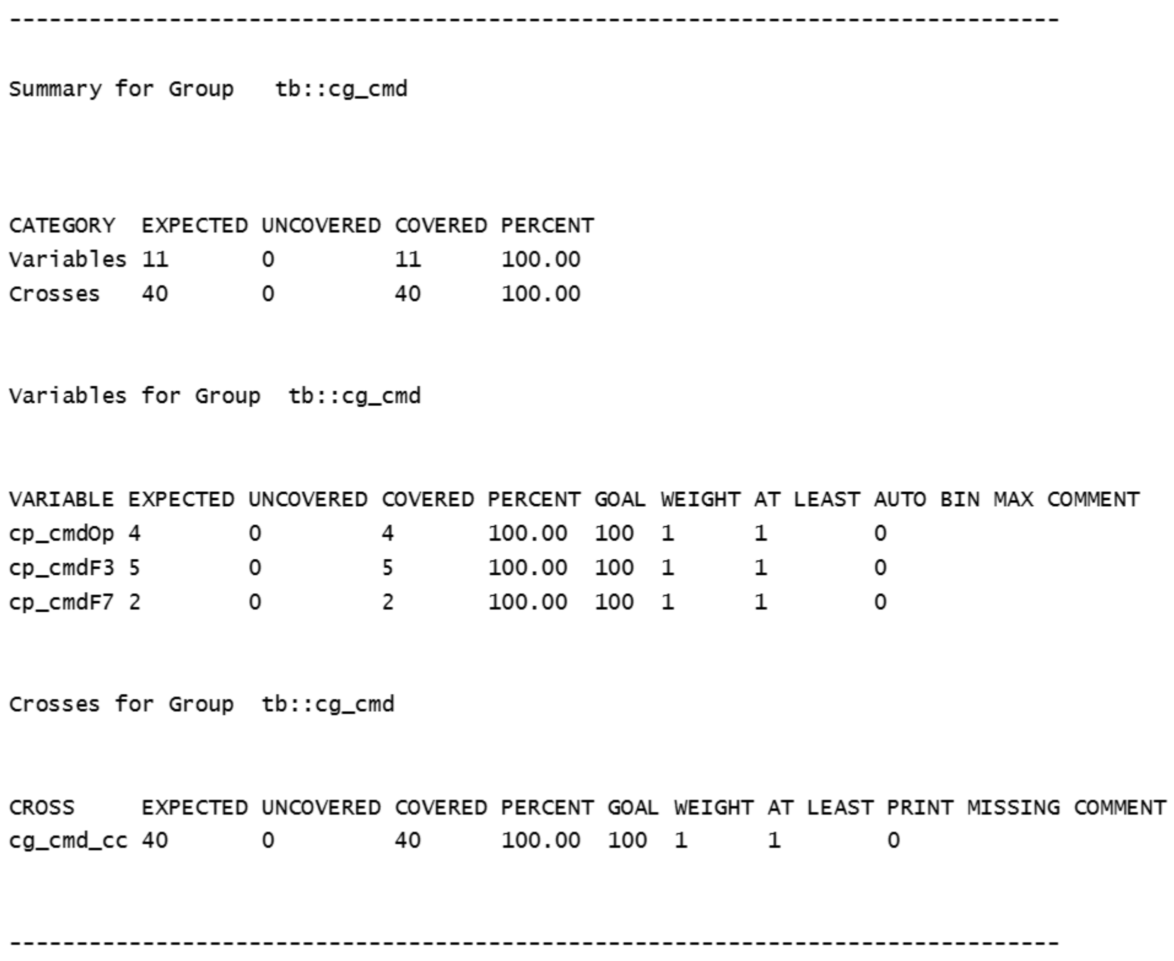
```
Bins

cp_addr       cp_data  COUNT AT LEAST
all_addrs_00 all_data 4      1
all_addrs_01 all_data 8      1
all_addrs_02 all_data 1      1
all_addrs_03 all_data 7      1
all_addrs_04 all_data 3      1
all_addrs_05 all_data 2      1
all_addrs_06 all_data 2      1
all_addrs_07 all_data 5      1
all_addrs_08 all_data 6      1
all_addrs_09 all_data 6      1
all_addrs_0a all_data 1      1
all_addrs_0b all_data 1      1
all_addrs_0c all_data 3      1
all_addrs_0d all_data 6      1
all_addrs_0e all_data 3      1
all_addrs_0f all_data 4      1
all_addrs_10 all_data 5      1
all_addrs_11 all_data 1      1
all_addrs_12 all_data 4      1
all_addrs_13 all_data 2      1
all_addrs_14 all_data 7      1
all_addrs_15 all_data 1      1
all_addrs_16 all_data 4      1
all_addrs_17 all_data 5      1
all_addrs_18 all_data 5      1
all_addrs_19 all_data 5      1
all_addrs_1a all_data 6      1
all_addrs_1b all_data 3      1
all_addrs_1c all_data 1      1
all_addrs_1d all_data 3      1
all_addrs_1e all_data 7      1
all_addrs_1f all_data 7      1
```

```
Tests

Total Coverage Summary
SCORE  GROUP
100.00 100.00


Total tests in report: 1
---------------------------
```

Pass/Fail Data:

```
PASS: Checking read data from register 3. Expected: 1499859087, Actual: 1499859087
PASS: Checking read data from register 22. Expected: 100214679, Actual: 100214679
PASS: Checking read data from register 1. Expected: 1737465566, Actual: 1737465566
PASS: Checking read data from register 9. Expected: 1857205839, Actual: 1857205839
PASS: Checking read data from register 13. Expected: 2817489823, Actual: 2817489823
PASS: Checking read data from register 7. Expected: 736026376, Actual: 736026376
PASS: Checking read data from register 16. Expected: 684579169, Actual: 684579169
PASS: Checking read data from register 18. Expected: 3475812885, Actual: 3475812885
PASS: Checking read data from register 29. Expected: 1013520973, Actual: 1013520973
PASS: Checking read data from register 26. Expected: 3975994486, Actual: 3975994486
PASS: Checking read data from register 15. Expected: 2450899489, Actual: 2450899489
PASS: Checking read data from register 31. Expected: 186429564, Actual: 186429564
PASS: Checking read data from register 31. Expected: 920350938, Actual: 920350938
PASS: Checking read data from register 30. Expected: 1190101064, Actual: 1190101064
PASS: Checking read data from register 26. Expected: 1906792980, Actual: 1906792980
PASS: Checking read data from register 27. Expected: 3163433420, Actual: 3163433420
PASS: Checking read data from register 20. Expected: 301164408, Actual: 301164408
PASS: Checking read data from register 3. Expected: 2217967762, Actual: 2217967762
PASS: Checking read data from register 8. Expected: 3853776915, Actual: 3853776915
PASS: Checking read data from register 6. Expected: 1984623650, Actual: 1984623650
PASS: Checking read data from register 0. Expected: 0, Actual: 0
PASS: Checking read data from register 17. Expected: 3380133878, Actual: 3380133878
PASS: Checking read data from register 8. Expected: 2353491458, Actual: 2353491458
PASS: Checking read data from register 29. Expected: 2844104390, Actual: 2844104390
PASS: Checking read data from register 13. Expected: 2727041488, Actual: 2727041488
PASS: Checking read data from register 31. Expected: 599557697, Actual: 599557697
PASS: Checking read data from register 15. Expected: 1307734997, Actual: 1307734997
PASS: Checking read data from register 1. Expected: 688604663, Actual: 688604663
PASS: Checking read data from register 25. Expected: 2635556379, Actual: 2635556379
PASS: Checking read data from register 8. Expected: 1199312603, Actual: 1199312603
PASS: Checking read data from register 30. Expected: 2761938464, Actual: 2761938464
PASS: Checking read data from register 23. Expected: 2709567306, Actual: 2709567306
```

Control:
Coverage Data:



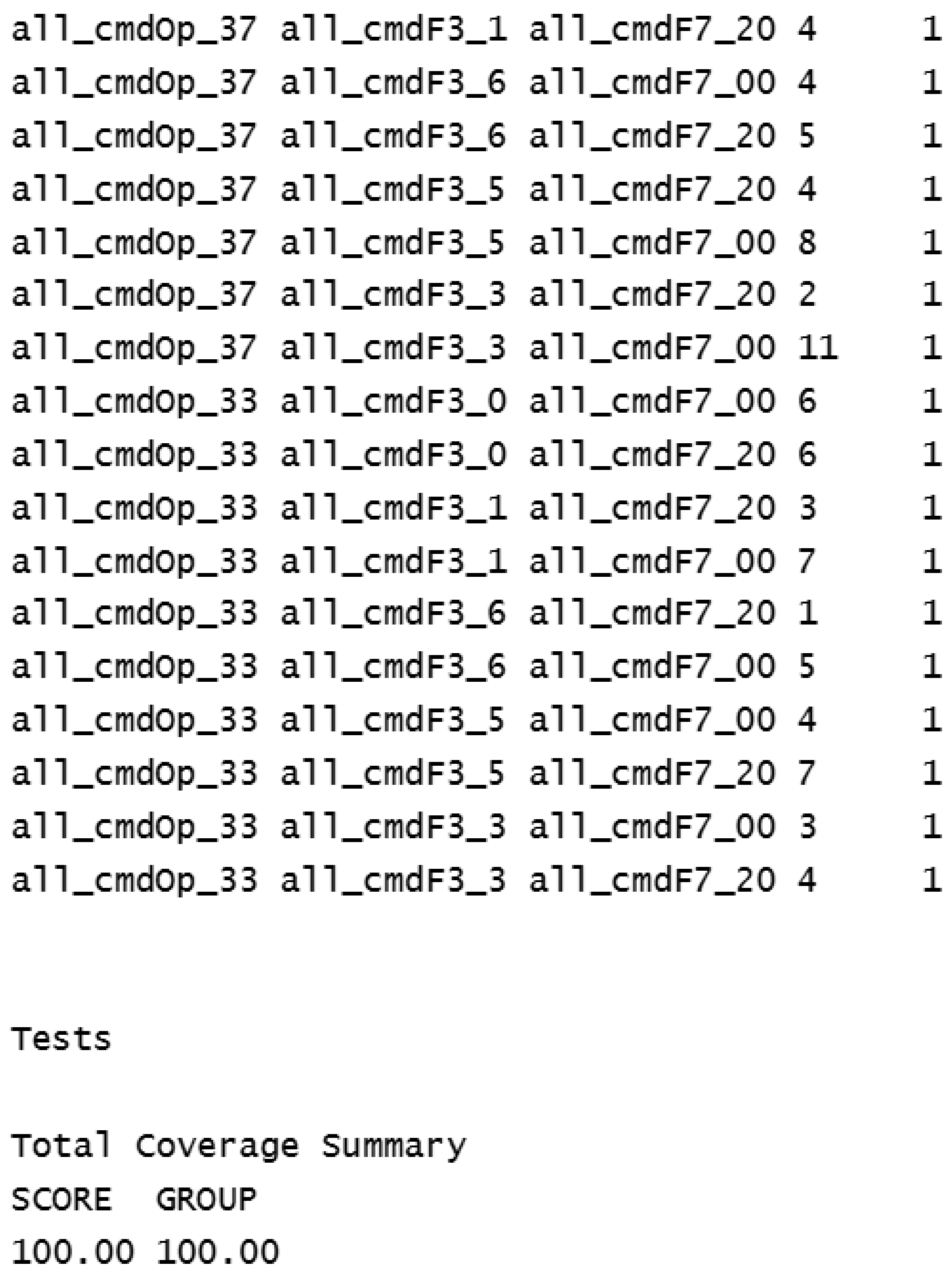
```
--------------------------------------------------------------------------------

Summary for Group   tb::cg_cmd



CATEGORY  EXPECTED UNCOVERED COVERED PERCENT
Variables 11       0         11      100.00
Crosses   40       0         40      100.00


Variables for Group  tb::cg_cmd


VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST AUTO BIN MAX COMMENT
cp_cmdOp 4        0         4       100.00  100  1      1        0
cp_cmdF3 5        0         5       100.00  100  1      1        0
cp_cmdF7 2        0         2       100.00  100  1      1        0


Crosses for Group  tb::cg_cmd


CROSS     EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST PRINT MISSING COMMENT
cg_cmd_cc 40       0         40      100.00  100  1      1        0


--------------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------------

Summary for Variable cp_cmdOp


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 4        0         4       100.00


User Defined Bins for cp_cmdOp


Bins

NAME         COUNT AT LEAST
all_cmdOp_13 130   1
all_cmdOp_63 122   1
all_cmdOp_37 121   1
all_cmdOp_33 127   1


---------------------------------------------------------------------------------

Summary for Variable cp_cmdF3


CATEGORY          EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 5        0         5       100.00


User Defined Bins for cp_cmdF3


Bins

NAME        COUNT AT LEAST
all_cmdF3_0 103   1
all_cmdF3_1 81    1
all_cmdF3_6 67    1
all_cmdF3_5 73    1
all_cmdF3_3 88    1


---------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------

Summary for Variable cp_cmdF7


CATEGORY           EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 2        0         2       100.00


User Defined Bins for cp_cmdF7


Bins

NAME         COUNT AT LEAST
all_cmdF7_00 133   1
all_cmdF7_20 123   1


--------------------------------------------------------------------------------

Summary for Cross cg_cmd_cc


Samples crossed: cp_cmdOp cp_cmdF3 cp_cmdF7
CATEGORY                           EXPECTED UNCOVERED COVERED PERCENT MISSING
Automatically Generated Cross Bins 40       0         40      100.00


Automatically Generated Cross Bins for cg_cmd_cc
```

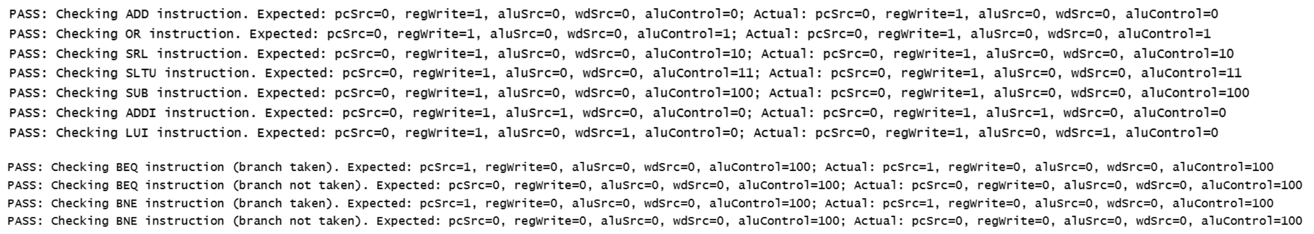
Bins

```
cp_cmdOp     cp_cmdF3    cp_cmdF7     COUNT AT LEAST
all_cmdOp_13 all_cmdF3_0 all_cmdF7_20 9     1
all_cmdOp_13 all_cmdF3_0 all_cmdF7_00 7     1
all_cmdOp_13 all_cmdF3_1 all_cmdF7_20 5     1
all_cmdOp_13 all_cmdF3_1 all_cmdF7_00 9     1
all_cmdOp_13 all_cmdF3_6 all_cmdF7_20 9     1
all_cmdOp_13 all_cmdF3_6 all_cmdF7_00 2     1
all_cmdOp_13 all_cmdF3_5 all_cmdF7_00 3     1
all_cmdOp_13 all_cmdF3_5 all_cmdF7_20 3     1
all_cmdOp_13 all_cmdF3_3 all_cmdF7_20 2     1
all_cmdOp_13 all_cmdF3_3 all_cmdF7_00 7     1
all_cmdOp_63 all_cmdF3_0 all_cmdF7_00 7     1
all_cmdOp_63 all_cmdF3_0 all_cmdF7_20 8     1
all_cmdOp_63 all_cmdF3_1 all_cmdF7_00 1     1
all_cmdOp_63 all_cmdF3_1 all_cmdF7_20 9     1
all_cmdOp_63 all_cmdF3_6 all_cmdF7_20 4     1
all_cmdOp_63 all_cmdF3_6 all_cmdF7_00 4     1
all_cmdOp_63 all_cmdF3_5 all_cmdF7_00 2     1
all_cmdOp_63 all_cmdF3_5 all_cmdF7_20 3     1
all_cmdOp_63 all_cmdF3_3 all_cmdF7_20 6     1
all_cmdOp_63 all_cmdF3_3 all_cmdF7_00 8     1
all_cmdOp_37 all_cmdF3_0 all_cmdF7_00 6     1
all_cmdOp_37 all_cmdF3_0 all_cmdF7_20 2     1
all_cmdOp_37 all_cmdF3_1 all_cmdF7_00 10    1
```



```
all_cmdOp_37 all_cmdF3_1 all_cmdF7_20 4     1
all_cmdOp_37 all_cmdF3_6 all_cmdF7_00 4     1
all_cmdOp_37 all_cmdF3_6 all_cmdF7_20 5     1
all_cmdOp_37 all_cmdF3_5 all_cmdF7_20 4     1
all_cmdOp_37 all_cmdF3_5 all_cmdF7_00 8     1
all_cmdOp_37 all_cmdF3_3 all_cmdF7_20 2     1
all_cmdOp_37 all_cmdF3_3 all_cmdF7_00 11    1
all_cmdOp_33 all_cmdF3_0 all_cmdF7_00 6     1
all_cmdOp_33 all_cmdF3_0 all_cmdF7_20 6     1
all_cmdOp_33 all_cmdF3_1 all_cmdF7_20 3     1
all_cmdOp_33 all_cmdF3_1 all_cmdF7_00 7     1
all_cmdOp_33 all_cmdF3_6 all_cmdF7_20 1     1
all_cmdOp_33 all_cmdF3_6 all_cmdF7_00 5     1
all_cmdOp_33 all_cmdF3_5 all_cmdF7_00 4     1
all_cmdOp_33 all_cmdF3_5 all_cmdF7_20 7     1
all_cmdOp_33 all_cmdF3_3 all_cmdF7_00 3     1
all_cmdOp_33 all_cmdF3_3 all_cmdF7_20 4     1


Tests

Total Coverage Summary
SCORE  GROUP
100.00 100.00
```

Pass/Fail Data:

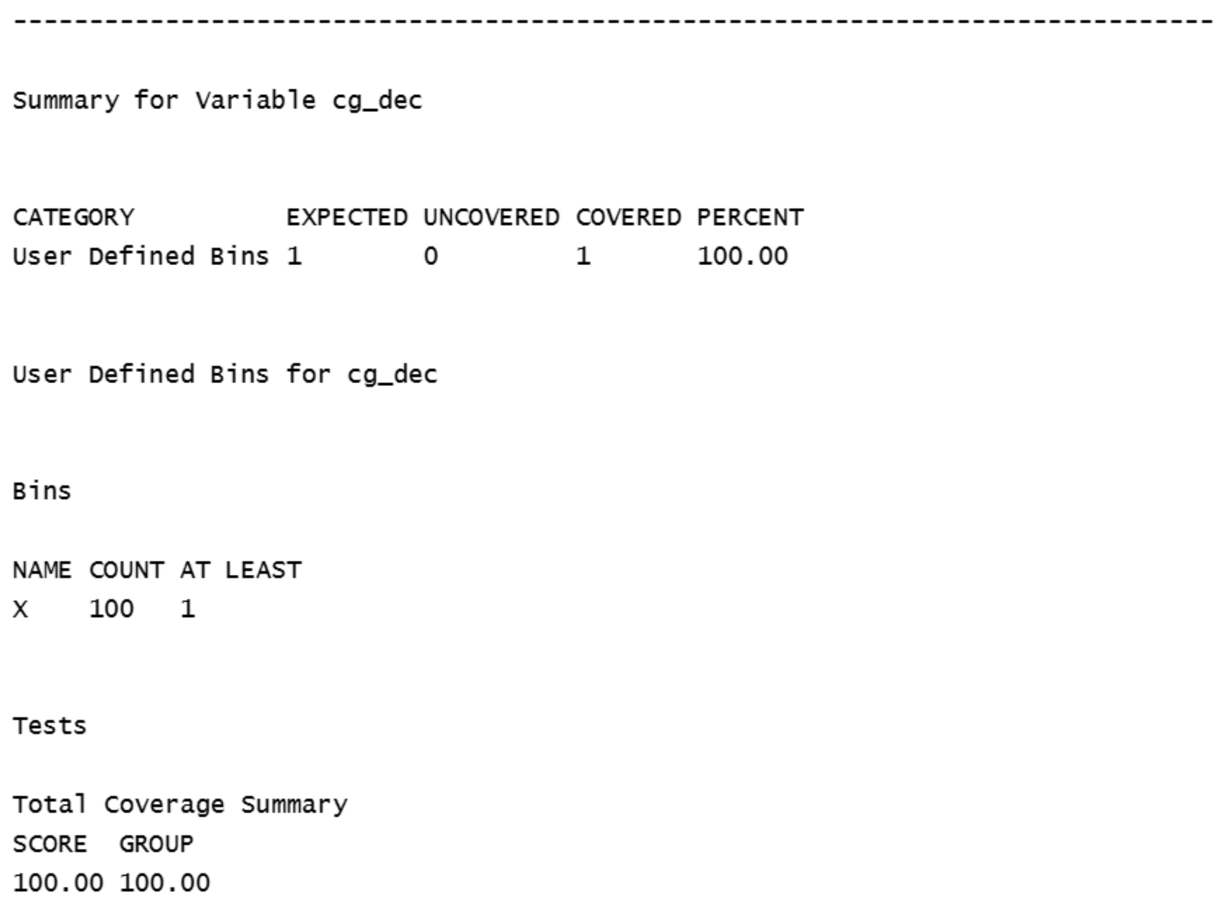


```
PASS: Checking ADD instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=0; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=0
PASS: Checking OR instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=1; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=1
PASS: Checking SRL instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=10; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=10
PASS: Checking SLTU instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=11; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=11
PASS: Checking SUB instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=100; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=0, aluControl=100
PASS: Checking ADDI instruction. Expected: pcSrc=0, regWrite=1, aluSrc=1, wdSrc=0, aluControl=0; Actual: pcSrc=0, regWrite=1, aluSrc=1, wdSrc=0, aluControl=0
PASS: Checking LUI instruction. Expected: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=1, aluControl=0; Actual: pcSrc=0, regWrite=1, aluSrc=0, wdSrc=1, aluControl=0

PASS: Checking BEQ instruction (branch taken). Expected: pcSrc=1, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100; Actual: pcSrc=1, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100
PASS: Checking BEQ instruction (branch not taken). Expected: pcSrc=0, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100; Actual: pcSrc=0, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100
PASS: Checking BNE instruction (branch taken). Expected: pcSrc=1, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100; Actual: pcSrc=1, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100
PASS: Checking BNE instruction (branch not taken). Expected: pcSrc=0, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100; Actual: pcSrc=0, regWrite=0, aluSrc=0, wdSrc=0, aluControl=100
```

Decoder:

Coverage Data:

```
/home/runner/DecodeTestbench.sv



-------------------------------------------------------------------------------

Summary for Group   tb::cg_decoder



CATEGORY  EXPECTED UNCOVERED COVERED PERCENT
Variables 1        0         1       100.00


Variables for Group  tb::cg_decoder


VARIABLE EXPECTED UNCOVERED COVERED PERCENT GOAL WEIGHT AT LEAST AUTO BIN MAX COMMENT
cg_dec   1        0         1       100.00  100  1      1        0


-------------------------------------------------------------------------------
```



```
--------------------------------------------------------------------------------

Summary for Variable cg_dec


CATEGORY           EXPECTED UNCOVERED COVERED PERCENT
User Defined Bins 1        0         1       100.00


User Defined Bins for cg_dec


Bins

NAME COUNT AT LEAST
X    100   1


Tests

Total Coverage Summary
SCORE  GROUP
100.00 100.00
```

Pass/Fail Data:

```
Input OpCode: 1111000011100110101011101100011
PASS: Checking the Instruction OpCode. Expected: 99, Actual: 99
Input OpCode: 10001100101100101100010110011
PASS: Checking the Instruction OpCode. Expected: 51, Actual: 51
Input OpCode: 101011001101010101100001100011
PASS: Checking the Instruction OpCode. Expected: 99, Actual: 99
Input OpCode: 11111111101001101111011001100011
PASS: Checking the Instruction OpCode. Expected: 99, Actual: 99
Input OpCode: 1001011010101000000000000010011
PASS: Checking the Instruction OpCode. Expected: 19, Actual: 19
Input OpCode: 11000111110110001010000110110111
PASS: Checking the Instruction OpCode. Expected: 55, Actual: 55
Input OpCode: 11100010001100001011011100010011
PASS: Checking the Instruction OpCode. Expected: 19, Actual: 19
Input OpCode: 10001100101010111011010000110111
PASS: Checking the Instruction OpCode. Expected: 55, Actual: 55
Input OpCode: 1000000101000101100111010110011
PASS: Checking the Instruction OpCode. Expected: 51, Actual: 51
Input OpCode: 10100001101011010011101100110011
PASS: Checking the Instruction OpCode. Expected: 51, Actual: 51
Input OpCode: 10011101110001011100100111100011
PASS: Checking the Instruction OpCode. Expected: 99, Actual: 99
```

```
I-Type Instruction: 4294124307
cmdOp is correct: 19
rd is correct: 19
cmdF3 is correct: 2
rs1 is correct: 6
rs2 is correct: 4294967295
----------------------------------------
B-Type Instruction: 2138211
cmdOp is correct: 99
cmdF3 is correct: 2
rs1 is correct: 1
rs2 is correct: 2
immB is correct: 0
----------------------------------------
U-Type Instruction: 305418423
cmdOp is correct: 55
cmdF3 is correct: 1
immU is correct: 305418240
```