CE6304 : Computer Architecture
Project #1
Branch Prediction

Under the supervision of: Prof. Benjamin Carrion Schaefer

Sanmati Marabad, sxm210368
Dhavalashri Prasad, dxp210085

Part 1
Branch Prediction

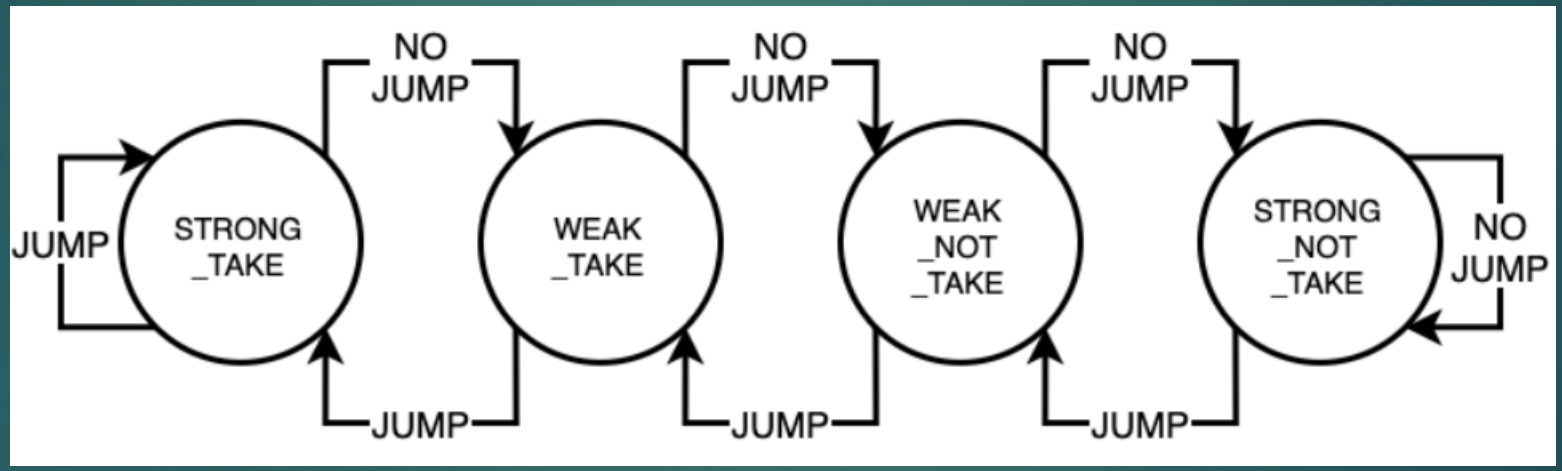
- Today's processors almost always employ the pipelining approach, which creates branch dangers when running programs. The hardware-level implementation of branch predictor aims to reduce the expenses of branching.
- The method entails only carrying out certain instructions when specific predicates are true. The processing of branch instructions is sped up by branch prediction.
- It enables the processor to begin executing instructions long before the branch outcome is certain.

The three branch predictor that are used are

1. Local Predictor
2. BiMode Predictor
3. Tournament Predictor

2 - bit Local Predictor

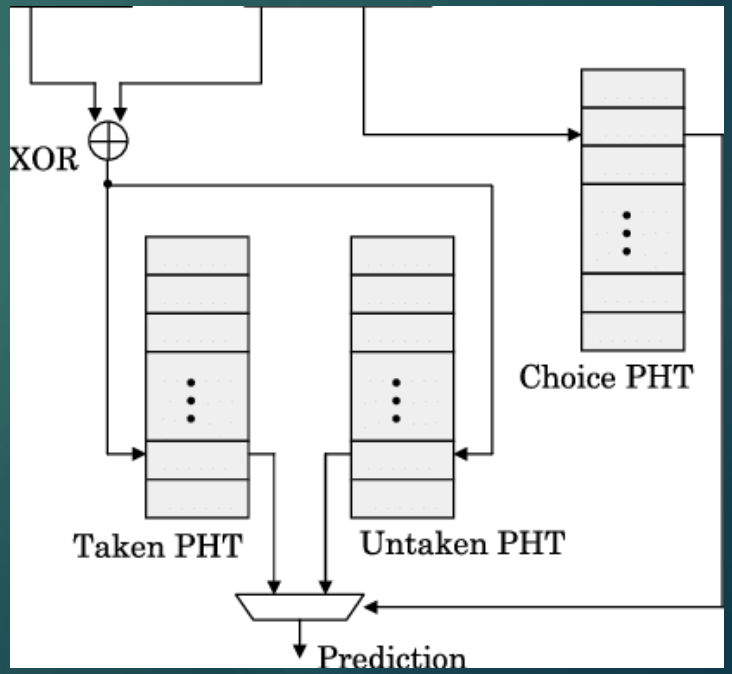
- In a local branch predictor, each conditional jump instruction has its own history buffer. Each conditional jump instruction has a separate history buffer, and the pattern history table may be independent for each instruction or shared by all conditional jumps.
- A 2 bit predictor simply adds another bit to the BHT to produce a state machine. First comes the prediction bit, followed by the hysteresis bit or conviction bit. Therefore, the 2-bit predictor should simply update its prediction after two false predictions.



Reference :
https://github.com/kuanying/two_bit_predictor_simulation/blob/main/images/branch_result.png

BiMode Predictor

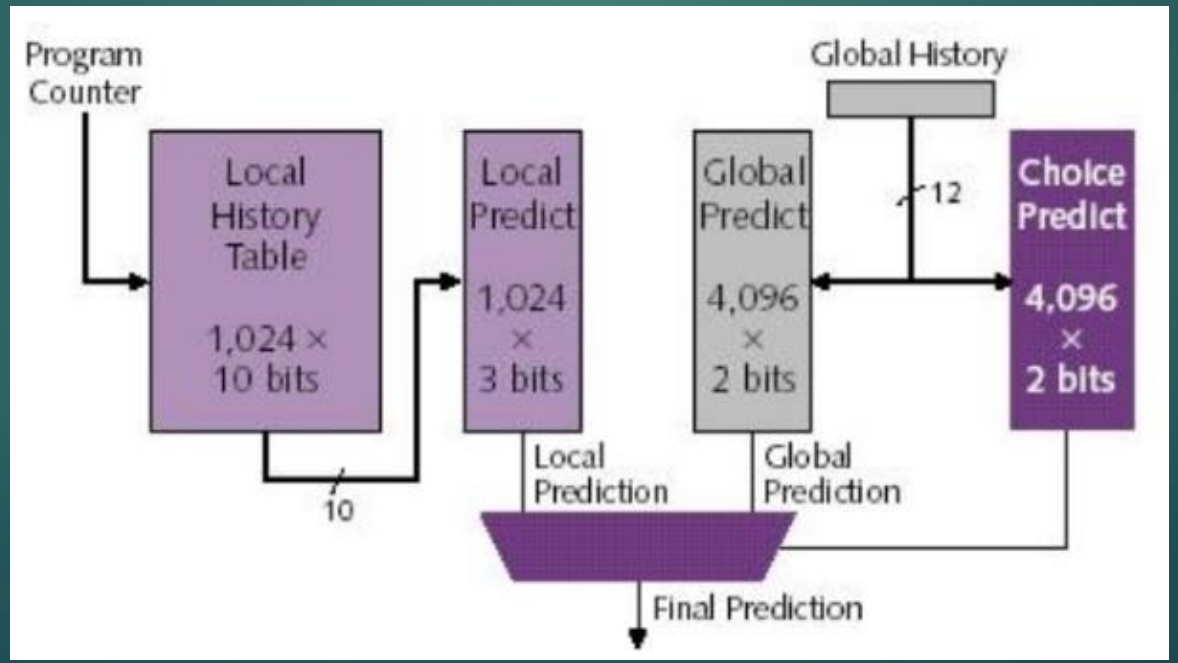
- The BiMode Predictor is a dynamic branch prediction approach that divides the prediction tables into two halves and dynamically chooses the appropriate half of the table for prediction based on the program's current mode. It is so simple that it has no effect on a processor's cycle time.
- A took array, a not-taken array, and a choice array are the three distinct history arrays that make up the bi-mode predictor, a two-level branch predictor. A hash of the PC and the entire history serves as the index for the taken/not-taken arrays. Only the PC has access to the choice array's index. The taken/not-taken arrays must be the same size because they share the same index.
- The destructive aliasing that happens when two branches with opposing biases share the same global history pattern is what the bi-mode branch predictor seeks to do away with. Destructive aliasing is minimized by dividing the predictors into taken/not-taken arrays and utilizing the branch's PC to choose between the two.



Reference : https://www.researchgate.net/figure/Bimode-plus-branch-predictor_fig2_421947

Tournament Predictor

- The Tournament Predictor has numerous predictions for each branch and a selection mechanism that determines which prediction to activate for a particular branch. Each branch index may have two predictions in a standard tournament predictor, one based on local information and the other on branch behavior globally.
- Which predictor to utilize for a particular prediction would be determined by a selector (Meta-Predictor). Predictors for PShare and GShare are trained on all branches. Whichever of the two predictors has been more accurate is preferred by the meta-predictor.
- The entry in the meta-predictor remains unchanged if both predictors are right or erroneous. The entry is increased if PShare is accurate but GShare is inaccurate. The entry is decreased if GShare is incorrect while PShare is accurate. PShare will be chosen by a higher Meta-Predictor entry, and GShare by a lower Meta-Predictor entry.



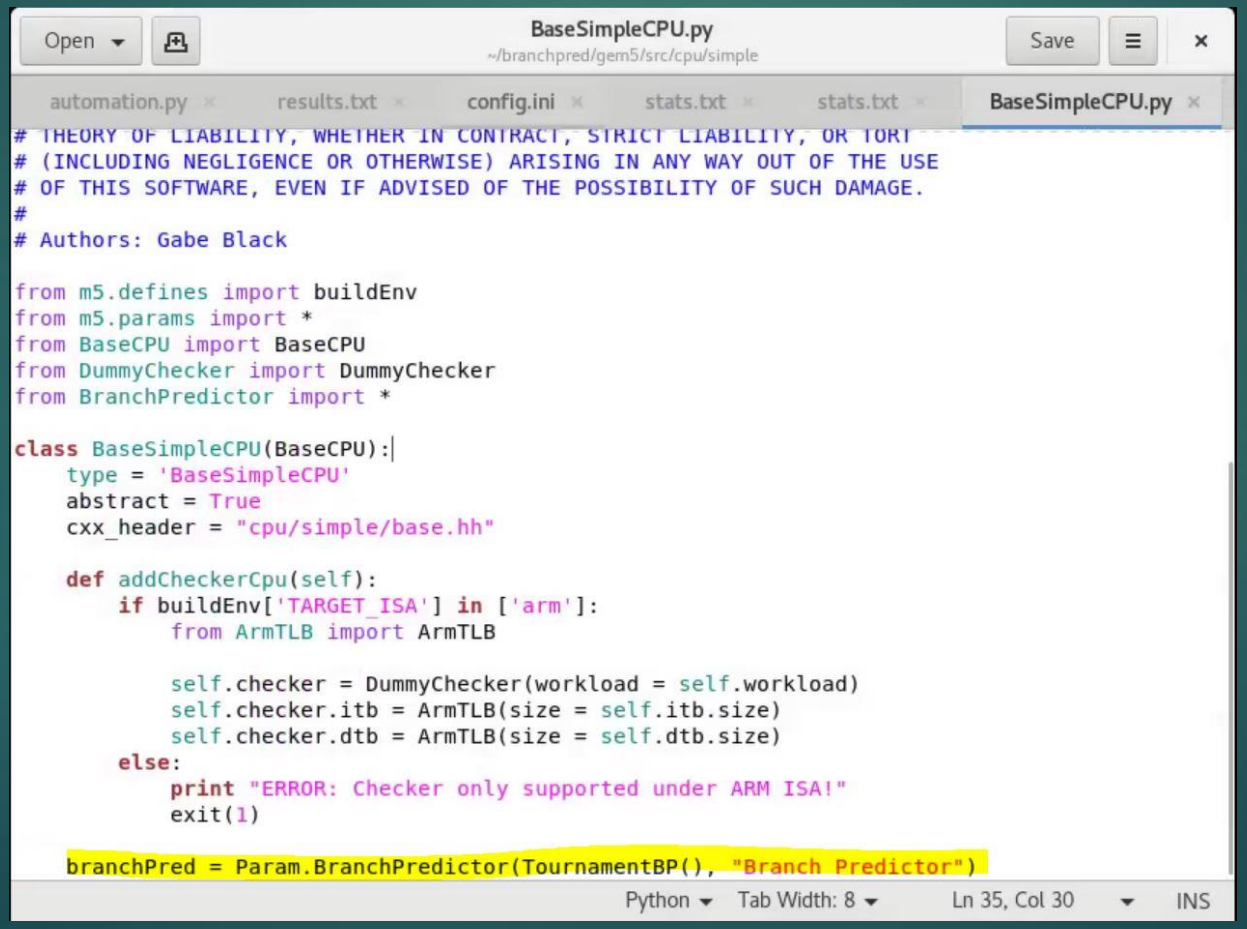
Gem5 - setup

- Setup done using UTD server (CE6304).
- Followed the steps according to the document shared.
- One of the problems faced was that the setup and execution took long time since multiple users were trying to access the same server.
- Path where the project is located in the server is

  /home/011/s/sx/sxm210368/branchpred/gem5

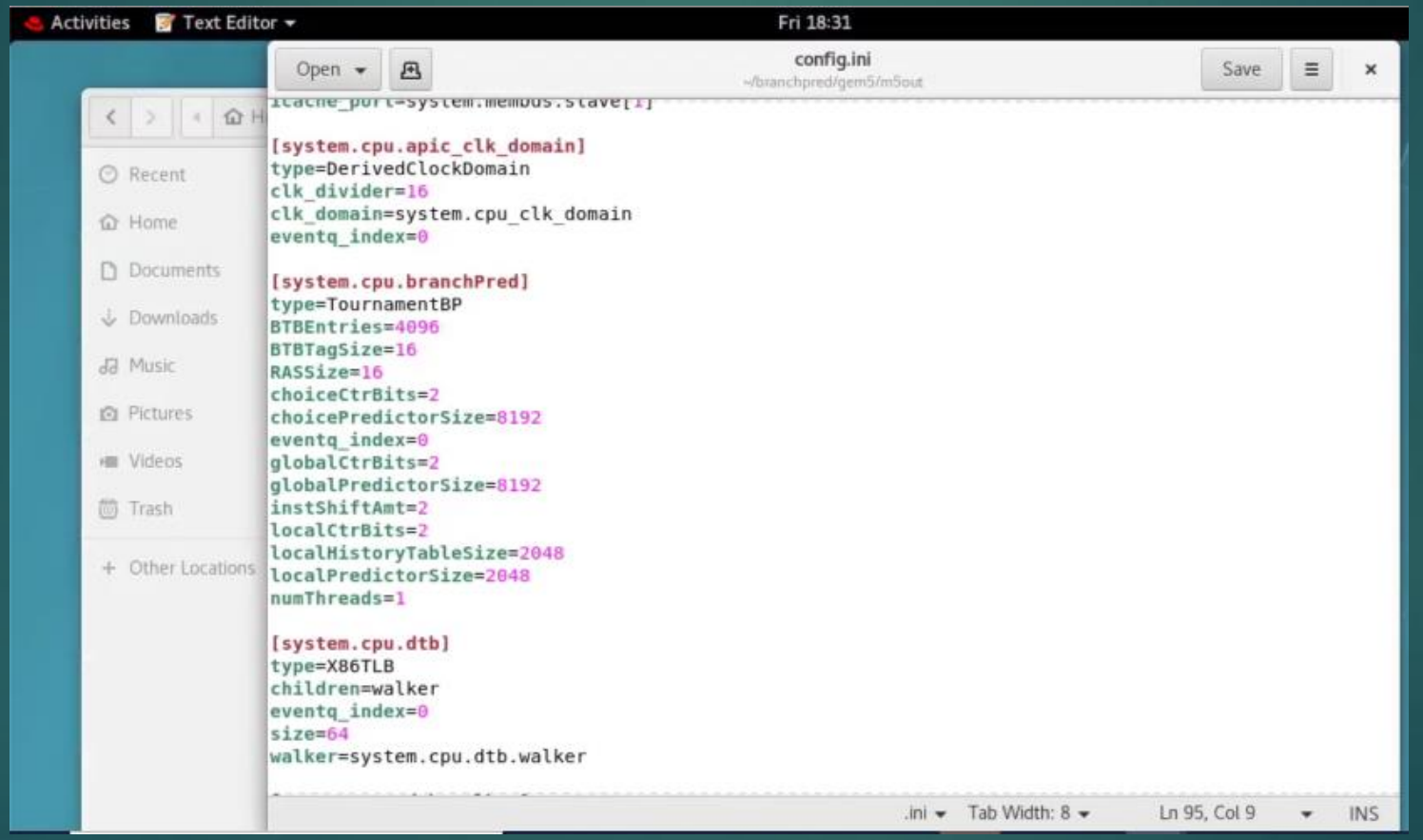
Part 2
Changing branch predictor

- In the file named “Basesimplecpu.py” the line where branch predictor has to be changed was updated to TournamentBP.



Result of changing Branch Predictor

config.ini



Part – 3

Changes made to add BTB Miss Percentage

File bpred_unit.hh

```
/** Stat for number of BTB hits. */
Stats::Scalar BTBHits;
/** Stat for number of times the BTB is correct. */
Stats::Scalar BTBCorrect;
/** Stat for percent times an entry in BTB found. */
Stats::Formula BTBHitPct;
/** Stat for percent times an entry in BTB is miss */
Stats::Formula BTBMissPct;
/** Stat for number of times the RAS is used to get a target. */
Stats::Scalar usedRAS;
/** Stat for number of times the RAS is incorrect. */
Stats::Scalar RASIncorrect;
```

File bpred_unit.cc

```
BTBHitPct
    .name(name() + ".BTBHitPct")
    .desc("BTB Hit Percentage")
    .precision(6);
BTBHitPct = (BTBHits / BTBLookups) * 100;

BTBMissPct
    .name(name() + ".BTBMissPct")
    .desc("BTB Miss Percentage")
    .precision(6);
BTBMissPct = (1-(BTBHits / BTBLookups)) * 100;

usedRAS
    .name(name() + ".usedRAS")
    .desc("Number of times the RAS was used to get a target.")
    ;
```

Stats.txt after adding BTBMissPct

```
system.cpu.branchPred.BTBCorrect                    0                       # Number of
correct BTB predictions (this stat may not work properly.
system.cpu.branchPred.BTBHitPct             90.744378                       # BTB Hit
Percentage
system.cpu.branchPred.BTBMissPct             9.255622                       # BTB Miss
Percentage
system.cpu.branchPred.usedRAS                  489567                       # Number of
times the RAS was used to get a target.
```

Changes made to add Branch MissPrediction Percentage

File exec_unit.hh

```
/// Number of branches predicted as taken
Stats::Scalar numPredictedBranches;
/// Number of mispredicted branches
Stats::Scalar numBranchMispred;
/// Percentage of mispredicted branches
Stats::Formula numBranchMispredPercent;
/// @}
```

File base.cc

```
            .prereq(t_info.numBranchMispred);

        t_info.numBranchMispredPercent
            .name(thread_str + ".numBranchMispredPercent")
            .desc("Branch misprediction percentage")
            .prereq(t_info.numBranchMispred);
        t_info.numBranchMispredPercent=(t_info.numBranchMispred/t_info.numBranc?
hes)*100;
    }
}
```

Result of adding Branch MissPrediction Percentage in stats.txt

```
system.cpu.BranchMispred                     3966                       # Number of branch
mispredictions
system.cpu.numBranchMispredPercent           0.382461                   # Branch
misprediction percentage
system.cpu.op_class::No_OpClass              26919       0.30%      0.30% # Class of
executed instruction
system.cpu.op_class::IntAlu                7050269      77.66%     77.96% # Class of
executed instruction
system.cpu.op_class::IntMult                 17682       0.19%     78.16% # Class of
```

Plain Text ▾ Tab Width: 8 ▾ Ln 315, Col 54 ▾ INS

Part – 4
Changes made to BranchPredictor.py

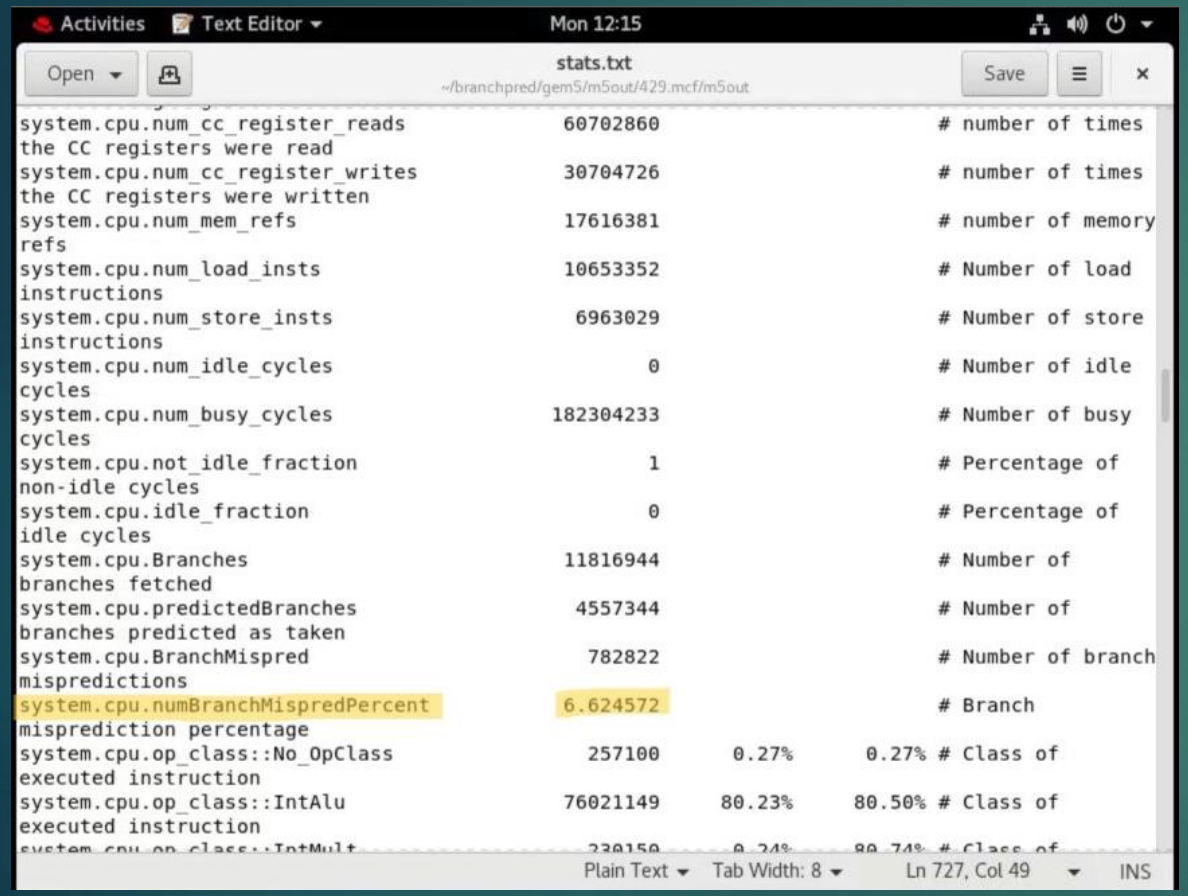
```
class BranchPredictor(SimObject):
    type = 'BranchPredictor'
    cxx_class = 'BPredUnit'
    cxx_header = "cpu/pred/bpred_unit.hh"
    abstract = True
    numThreads = Param.Unsigned(1, "Number of threads")
    BTBEntries = Param.Unsigned(2048, "Number of BTB entries")
    BTBTagSize = Param.Unsigned(16, "Size of the BTB tags, in bits")
    RASSize = Param.Unsigned(16, "RAS size")
    instShiftAmt = Param.Unsigned(2, "Number of bits to shift instructions by")

class LocalBP(BranchPredictor):
    type = 'LocalBP'
    cxx_class = 'LocalBP'
    cxx_header = "cpu/pred/2bit_local.hh"
    localPredictorSize = Param.Unsigned(1024, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
```

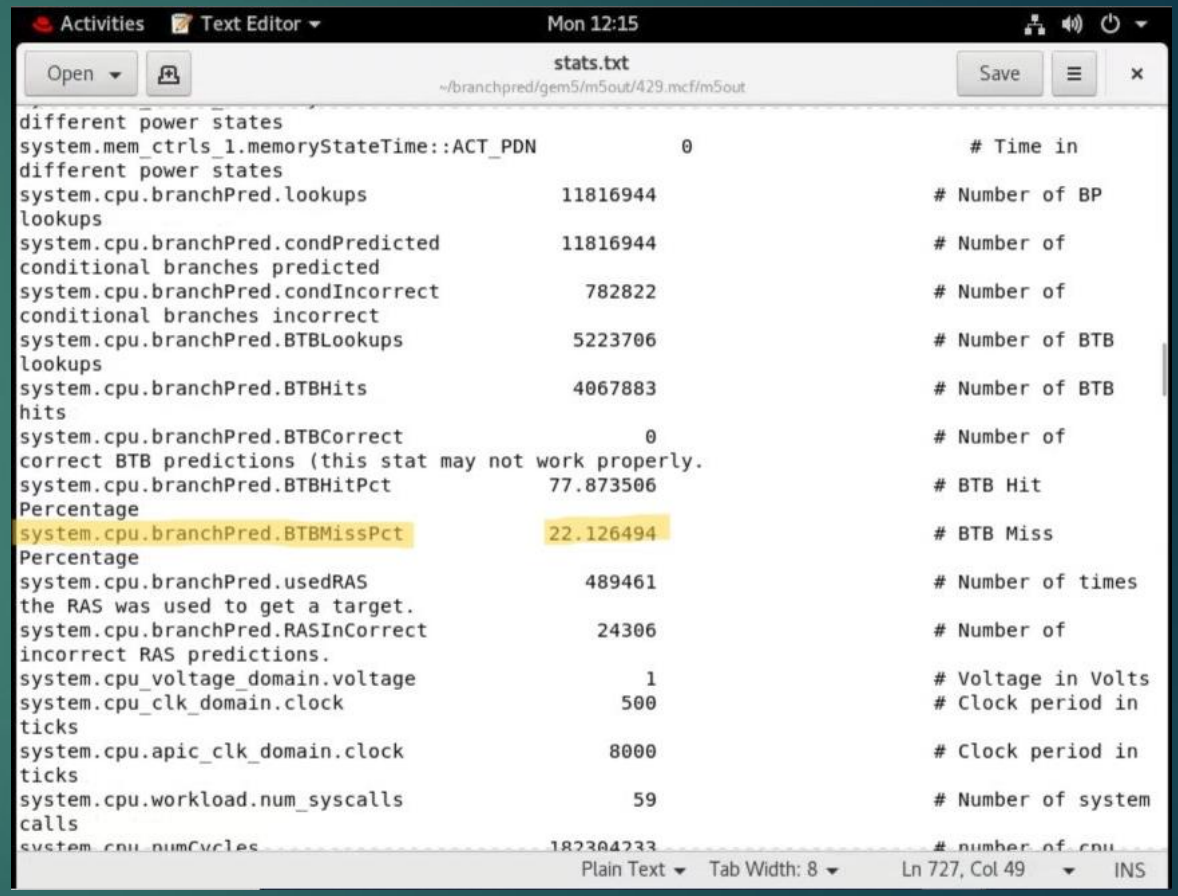
```
class TournamentBP(BranchPredictor):
    type = 'TournamentBP'
    cxx_class = 'TournamentBP'
    cxx_header = "cpu/pred/tournament.hh"
    localPredictorSize = Param.Unsigned(1024, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
    localHistoryTableSize = Param.Unsigned(2048, "size of local history table")
    globalPredictorSize = Param.Unsigned(4096, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(4096, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")


class BiModeBP(BranchPredictor):
    type = 'BiModeBP'
    cxx_class = 'BiModeBP'
    cxx_header = "cpu/pred/bi_mode.hh"
    globalPredictorSize = Param.Unsigned(2048, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(2048, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

Part 4 – Benchmark 429 LocalBP

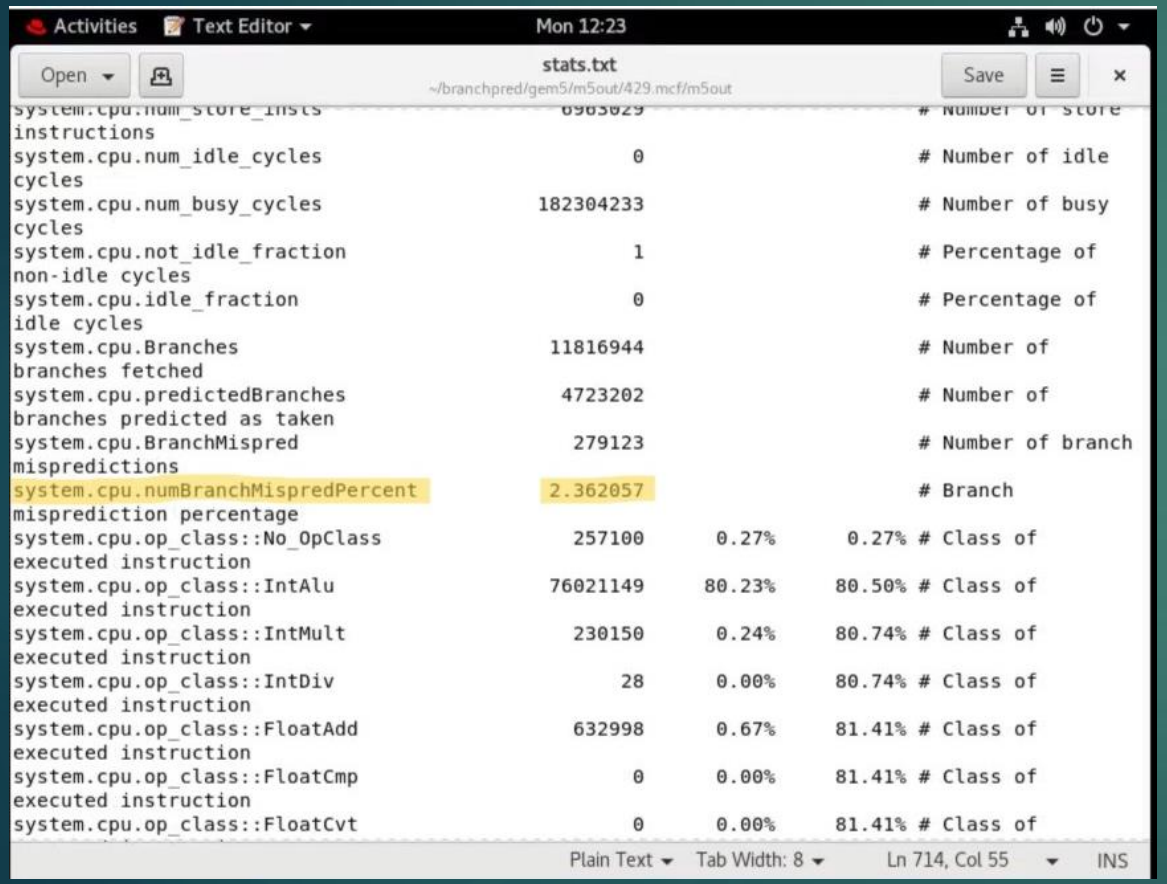


Branch misprediction percentage

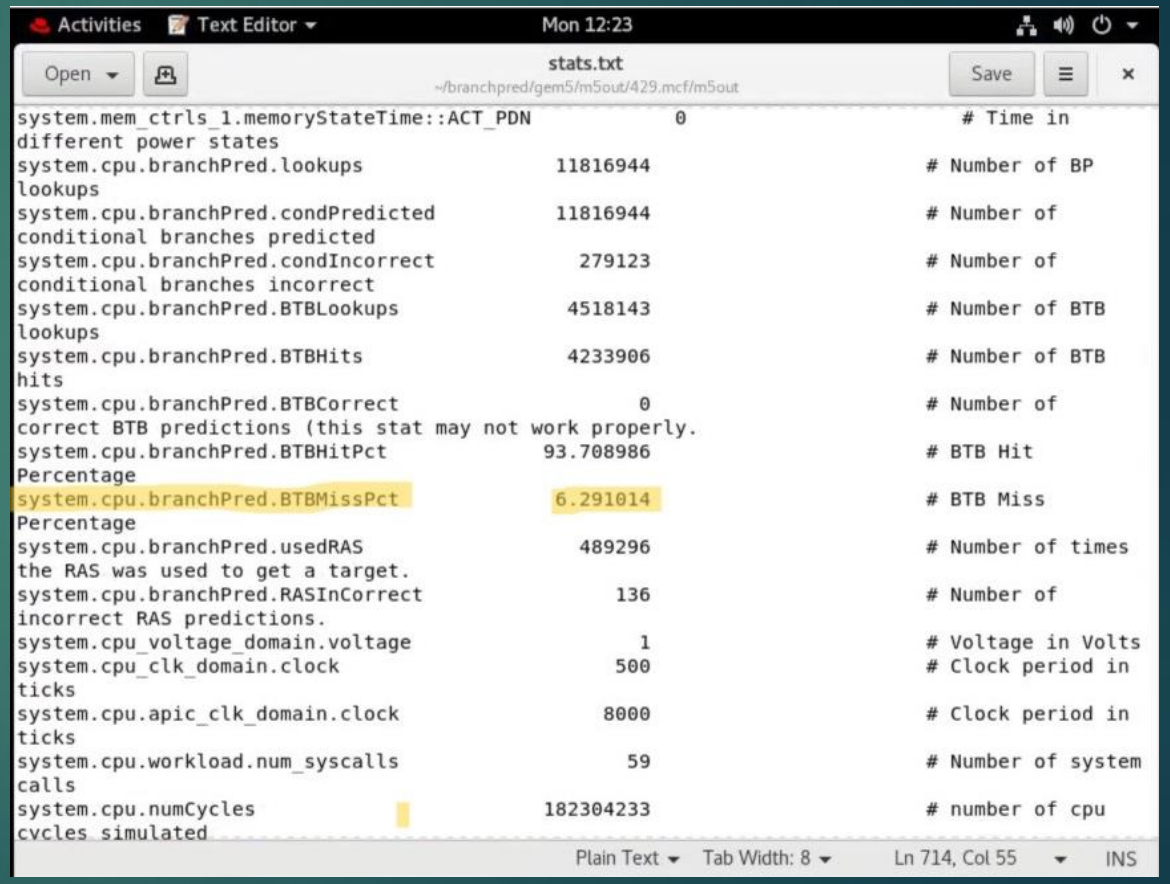


BTB Miss percentage

Part 4 – Benchmark 429 BiModeBP

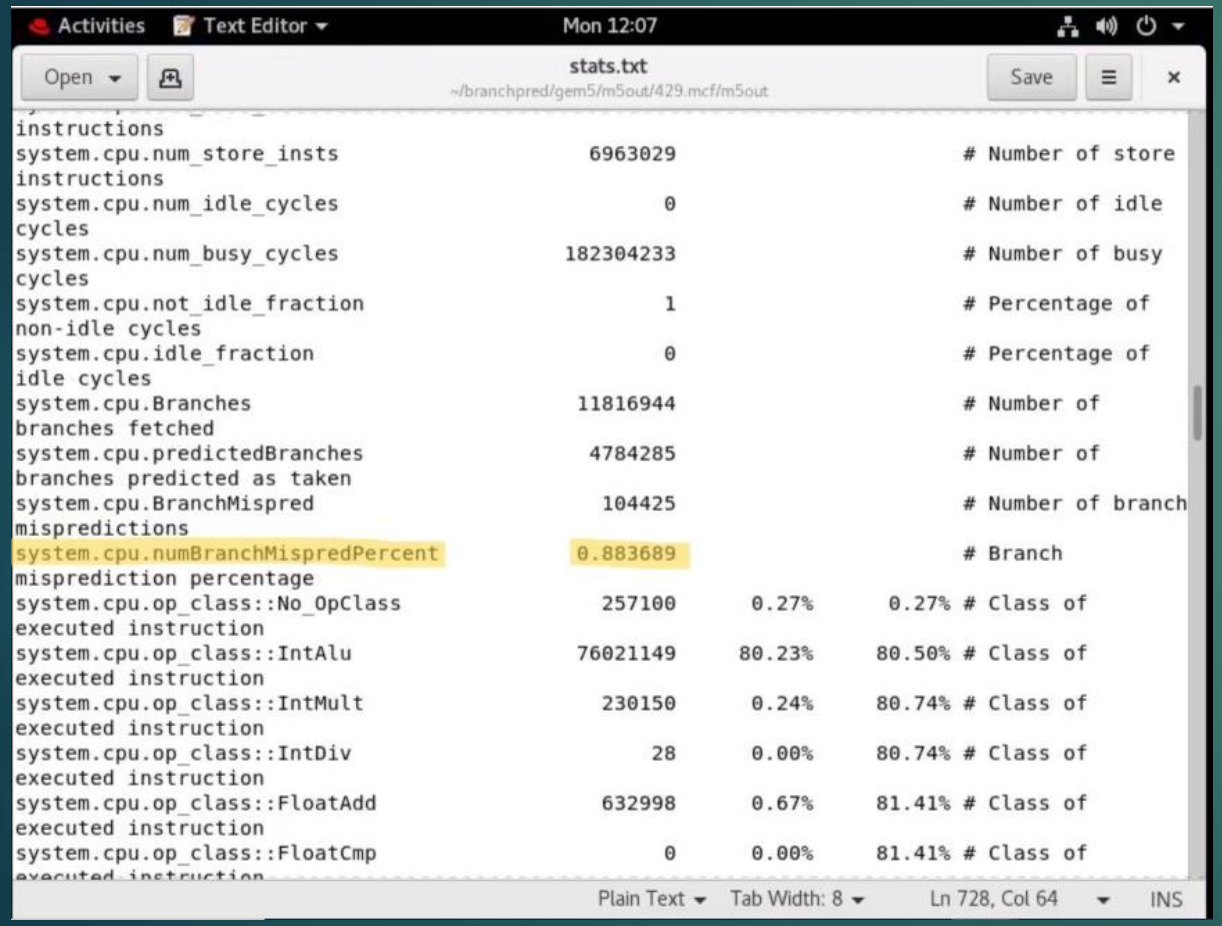


Branch misprediction percentage

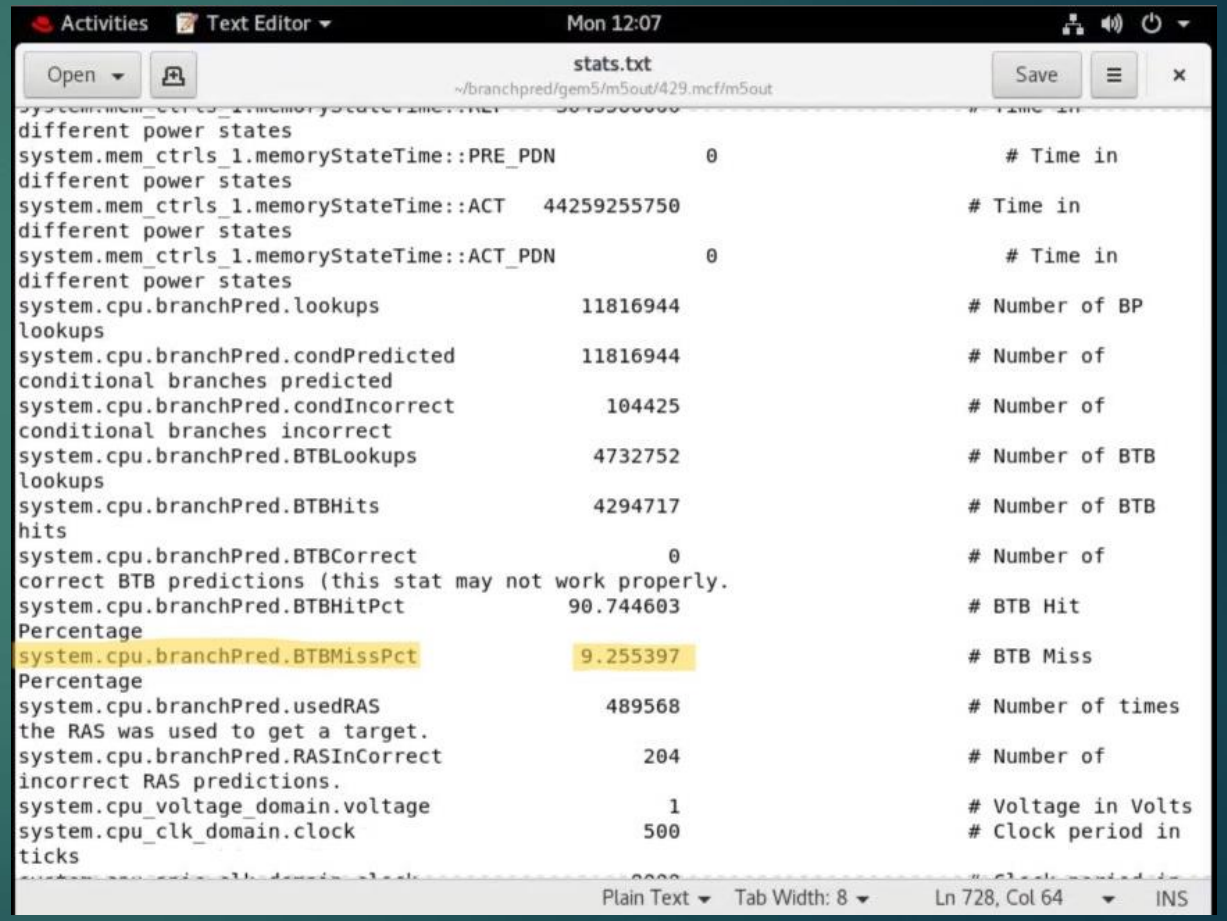


BTB Miss percentage

Part 4 – Benchmark 429 TournamentBP

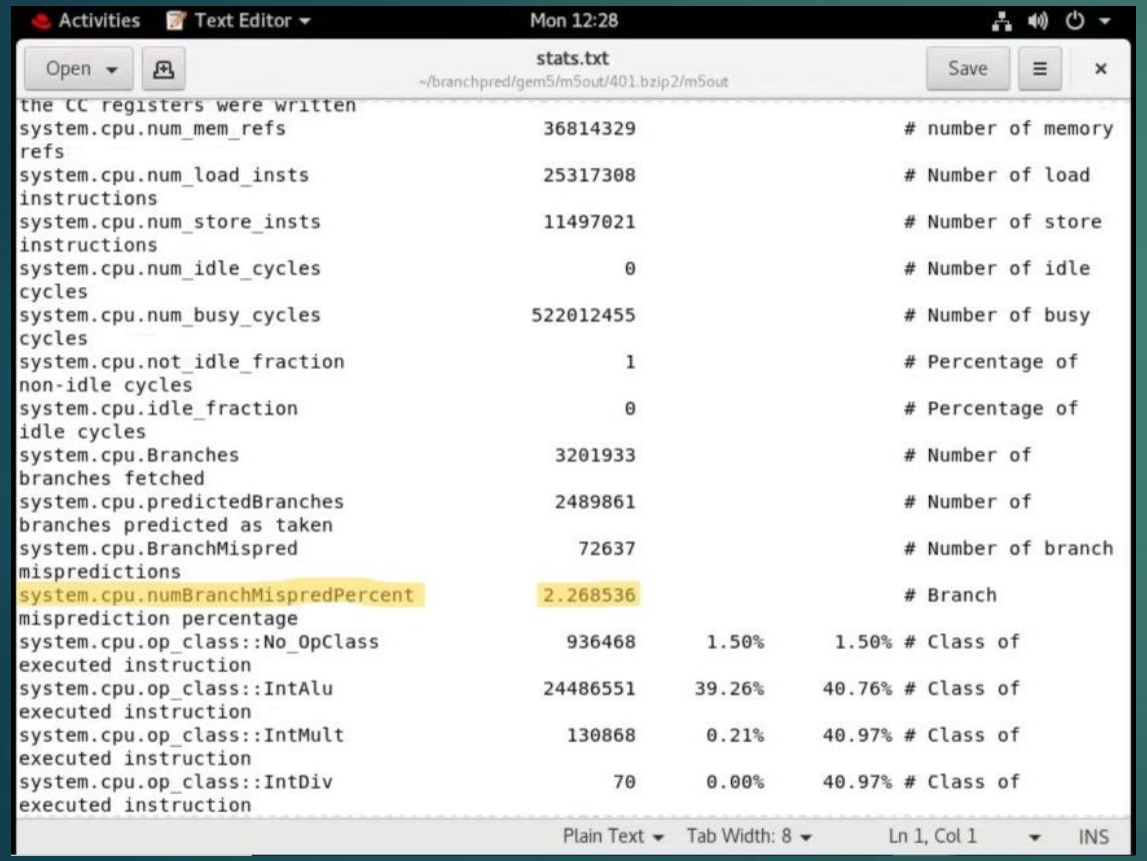


Branch misprediction percentage

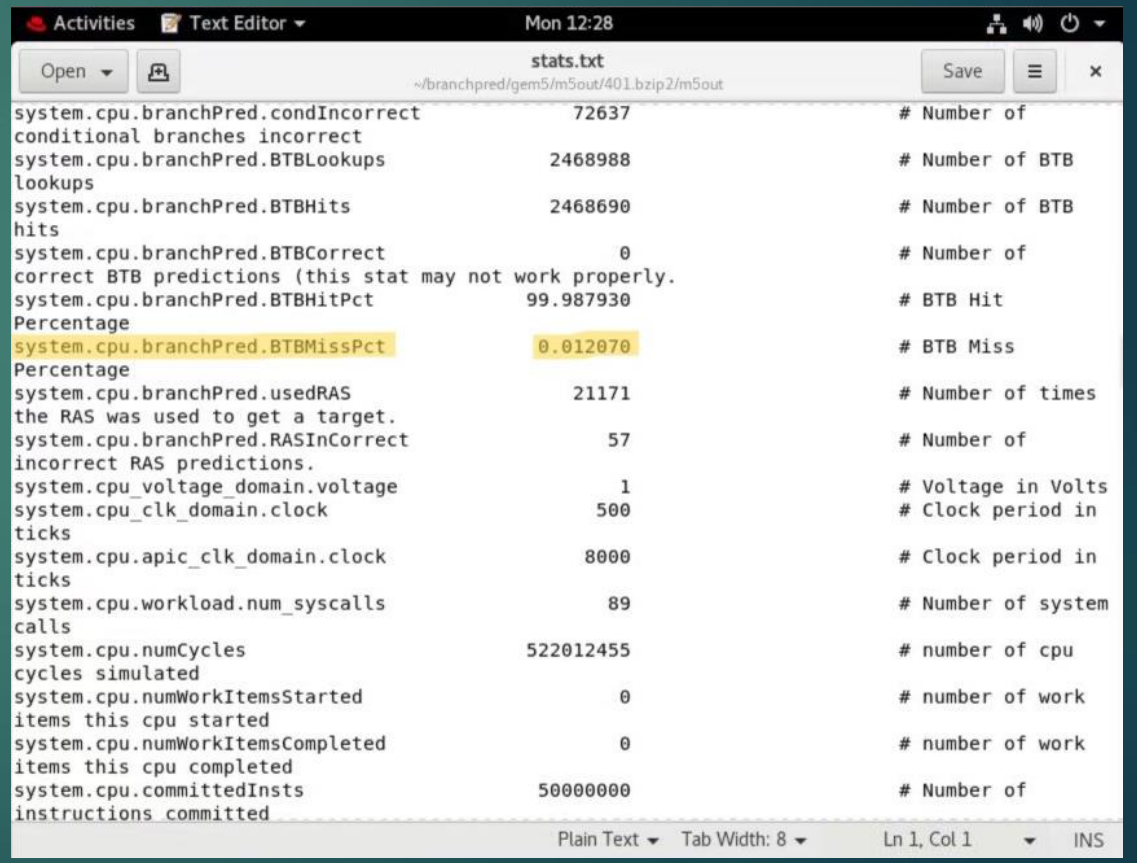


BTB Miss percentage

Part 4 – Benchmark 401 BiModeBP

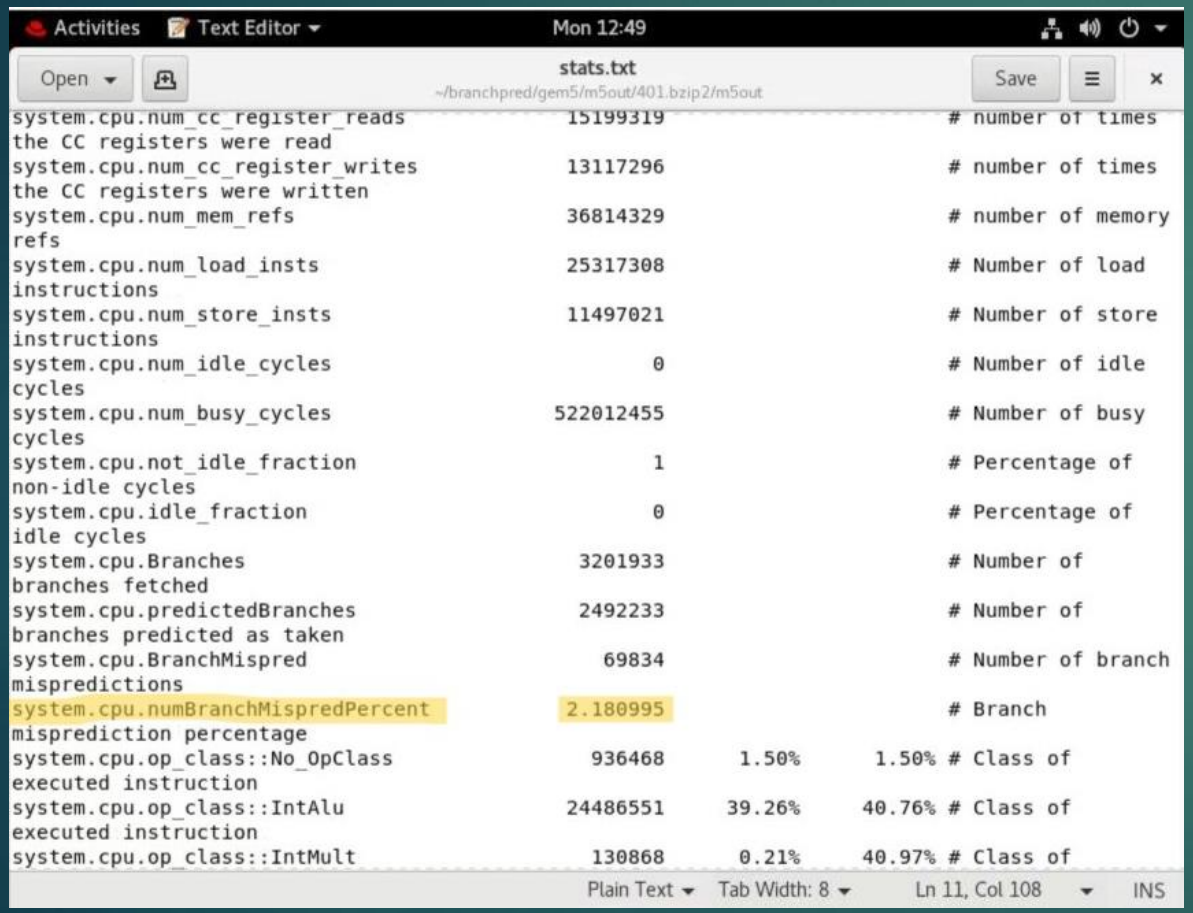


Branch misprediction percentage

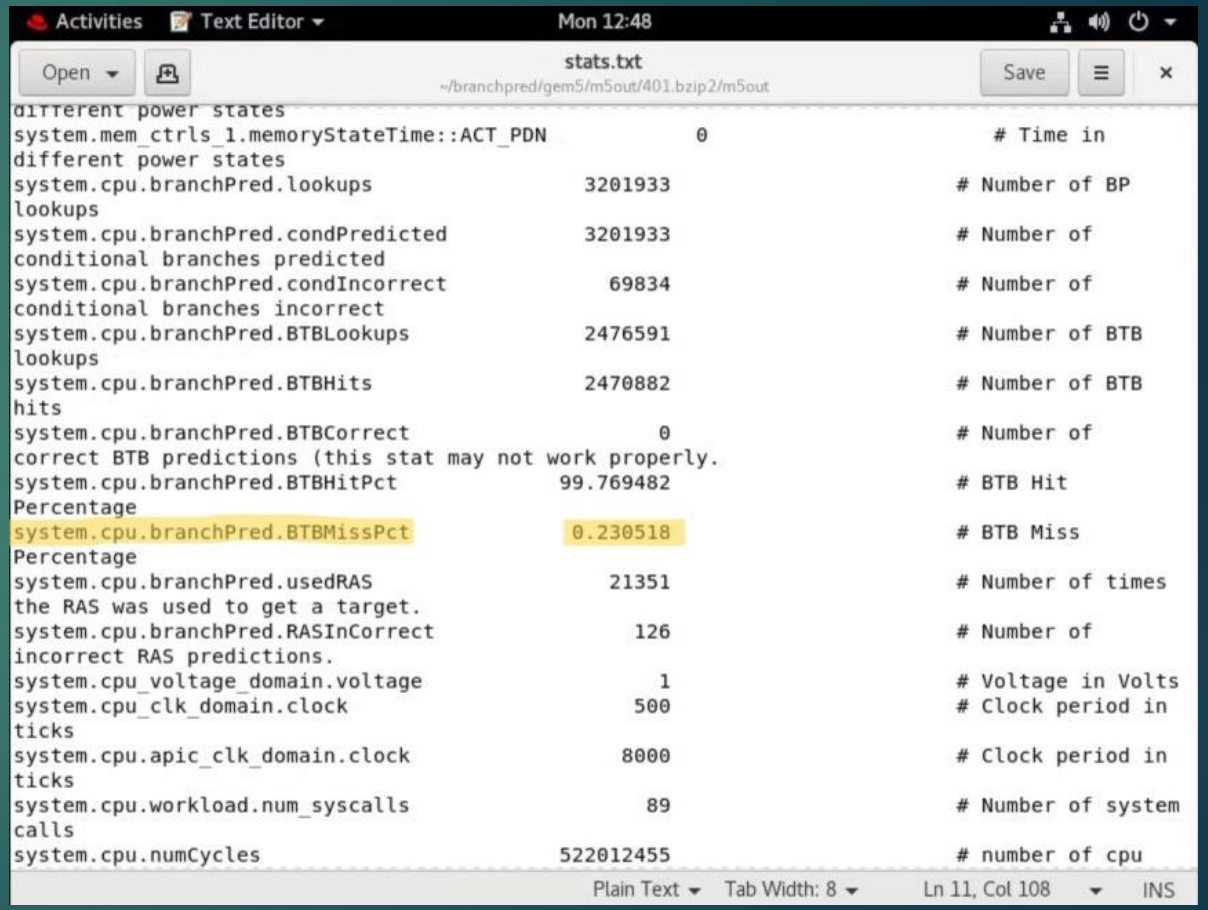


BTB Miss Percentage

Part 4 – Benchmark 401 TournamentBP

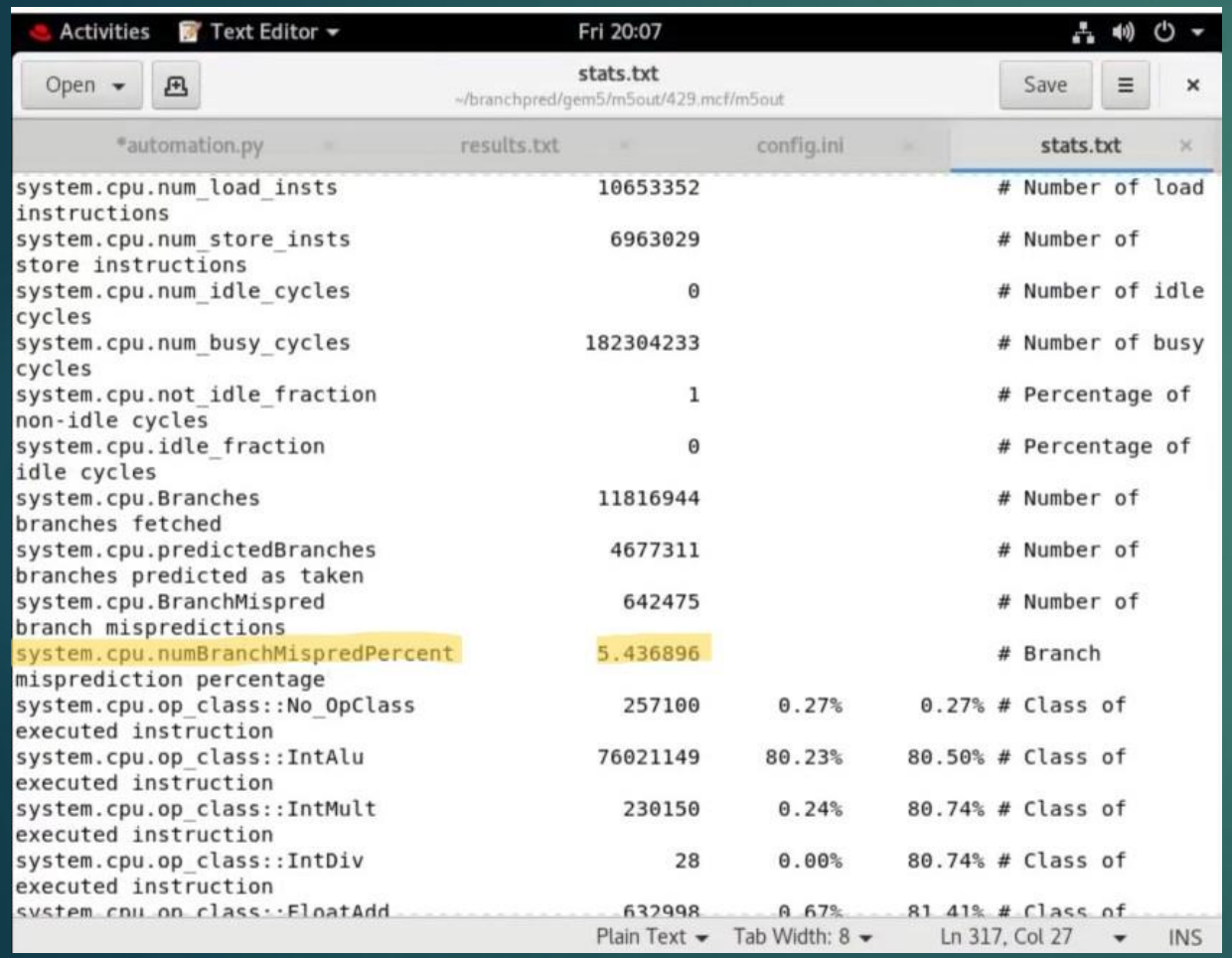


Branch misprediction percentage

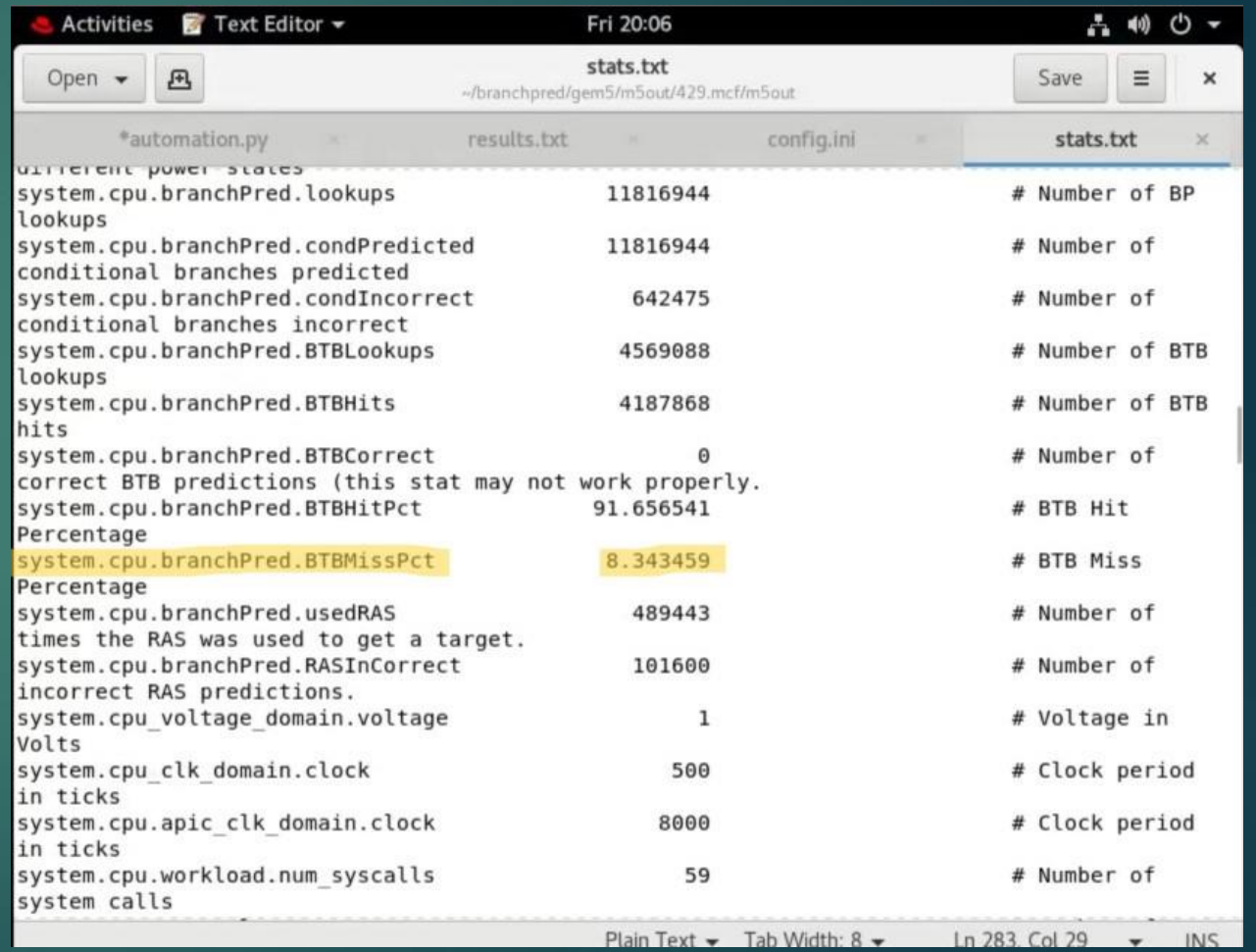


BTB Miss Percentage

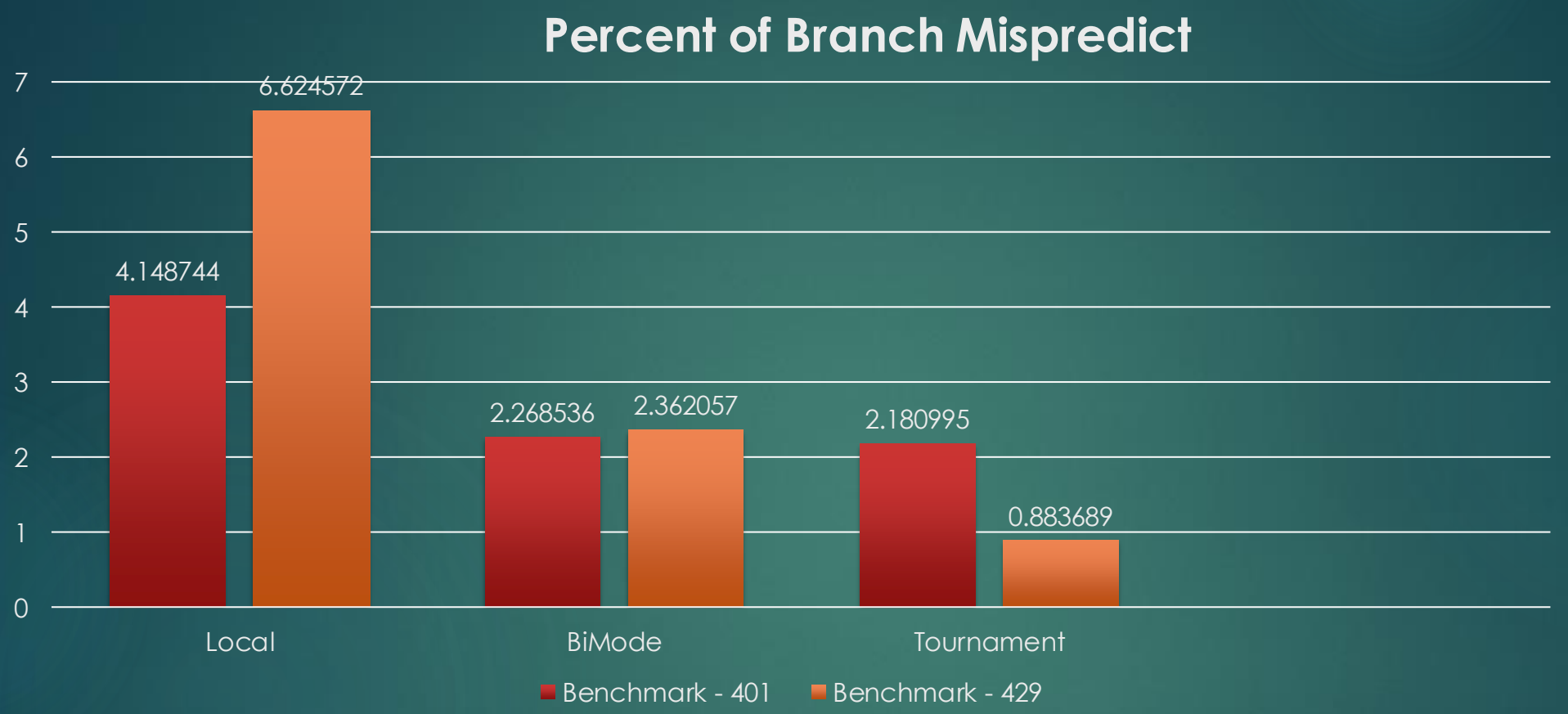
Part 4 – Benchmark 401 LocalBP



Branch Misprediction Percentage

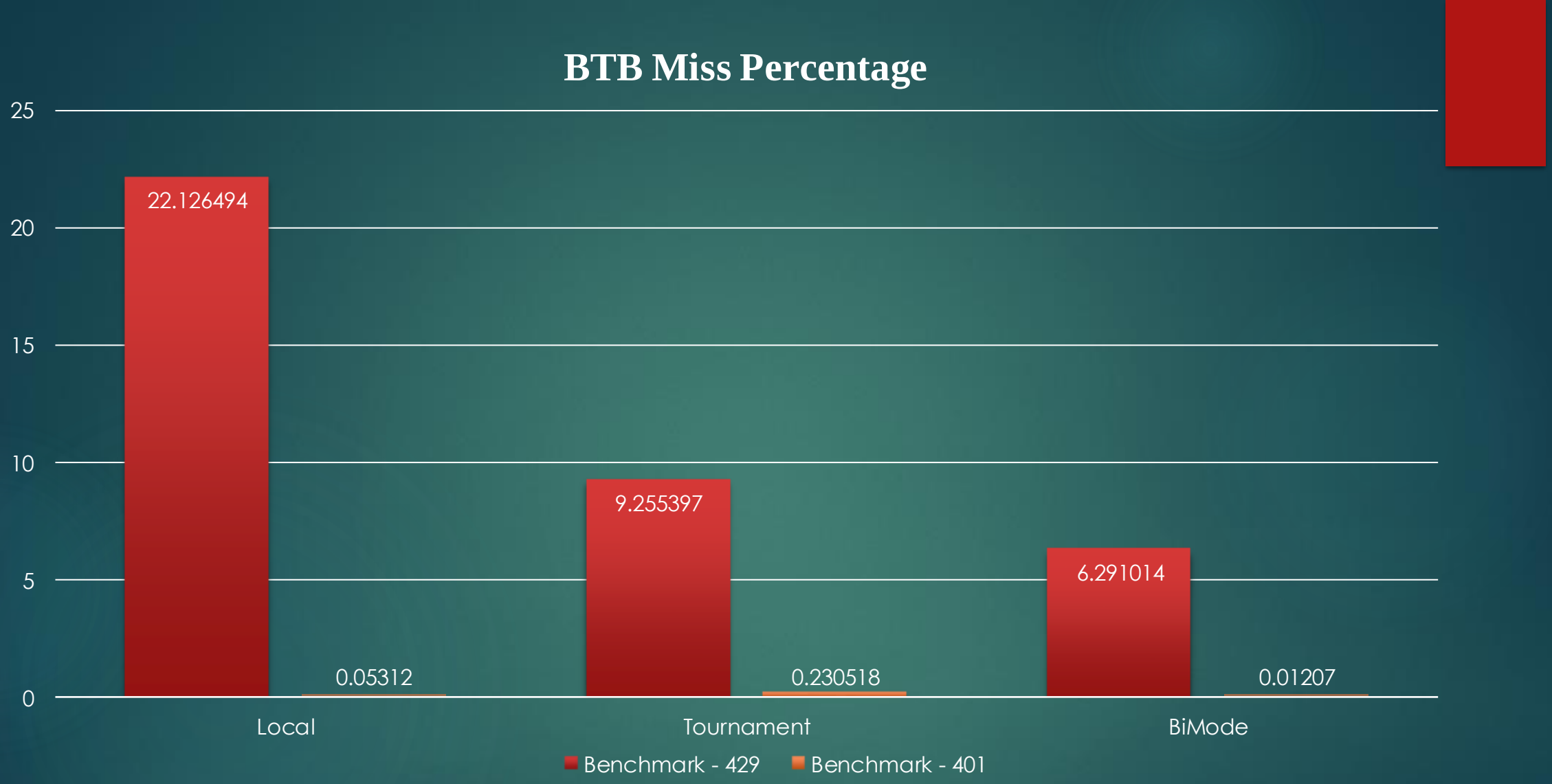


BTB Miss Percentage



TournamentBP has the lowest misprediction rate for both the benchmarks whereas LocalBP has the highest.

BTB Miss Percentage



BiModeBP has the lowest BTB Miss Pct for both the benchmarks.

Conclusion

- BiModeBP shows better branch misprediction percentage and BTB Miss percentage.

Automation

- Part – 2 and part – 4 of the project have been automated.

  File path

  /home/011/s/sx/sxm210368/branchpred/gem5/automation.py

- Results of the automation will be stored in results.txt

  File path

  /home/011/s/sx/sxm210368/branchpred/gem5/results.txt