Project 2 Instructions

Objective: Understand branch prediction functionality and implement branch prediction related functions in the gem5 simulator. A branch predictor (BP) can be implemented by defining relevant mechanisms through following functions: `lookup`, `update`, `uncondBranch`, `btbUpdate`, and `squash`. You can find the different functions about branch predictor from http://pages.cs.wisc.edu/~swilson/gem5-docs/classBPredUnit.html.

1. `lookup`: This function looks up a given branch address to index appropriate registers or counters and determines if the branch needs to be taken or not taken. If global history is used, it can be updated following the prediction. The parameters for the function are `branch_address` (to look up) and `bp_history` (pointer to a new object may be set, where the object contains shared state corresponding to the lookup for this branch). The method returns the prediction, i.e., whether this branch needs to be taken or not.
2. `update`: Based on the actual taken/not taken information, this function updates the BP including counters and global history associated with the branch.
   - `branch_address`: the branch's address that will be used to index appropriate counter.
   - `taken`: whether the branch is actually taken or not
   - `bp_history`: pointer to the shared state (aka global history) corresponding to this branch
   - `squashed`: it is set to true when there is a misprediction. If so, the counter's value should not be updated. Also, the global history can be restored back if it was updated during the prediction of the branch.
3. `uncondBranch`: This method is called when the branch instruction is an unconditional branch. Branch predictor will not do anything but update the global history with taken.
4. `btbUpdate`: This function is called only when there is a branch miss in BTB. This can happen when the BP does not know where to jump. In this case, the branch is explicitly predicted as not taken so that branch target address is not required. So, any updates to the global history needs to be modified to ensure prediction as not taken.
5. `squash`: This function squashes all outstanding updates until a given sequence number. Typically, the input parameter is a bp_history (Pointer to the global history object associated with this branch). The predictor will need to retrieve previous state and delete the object.

Understanding 2-bit local and Bi-mode branch predictors

The 2-bit local branch predictor is a simple branch predictor which has 4 states: strongly taken, weakly taken, weakly not taken, strongly not taken, denoted as (11, 10, 01, 00). You can find the implementation in the directory at `gem5/src/cpu/pred/` in the files 2bit_local.cc, 2-bit_local.hh, tournament.cc and tournament.hh.

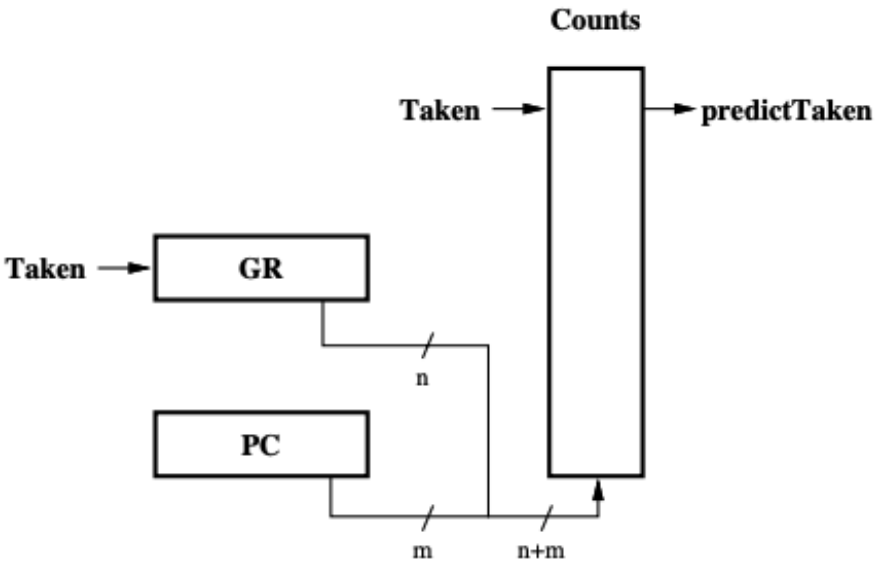
Explanation about branch predictors:

- 2b local BP: https://courses.cs.washington.edu/courses/csep548/06au/lectures/branchPred.pdf

- bi-mode BP: https://tnm.engin.umich.edu/wp-content/uploads/sites/353/2017/12/1997.12.Thebi-mode-branch-predictor.pdf (section 2.2, Figure 1)
- tournament BP: https://www.inf.ed.ac.uk/teaching/courses/car/Notes/2017-18/lecture05-handling_hazards.pdf

Understanding GSelect branch predictor

Read https://inst.eecs.berkeley.edu//~cs252/sp17/papers/McFarling-WRL-TN-36.pdf section 6, Fig. 8. "GSelect scheme is similar to bimodal predictor or branch history table. However, like correlation-based prediction, this method records history of branches into the shift register. Then, it concatenates (+ in the figure) 'n' bits from the branch history (GR) and 'm' bits from address of the branch instruction (PC). The result is used to index the table of prediction bits. Implementation can use either tagged or tagless tables."



NOTE: Before starting the project, please make sure your modifications in source code or configuration of parameters made during project 1 are reverted back. If not, please discard them or obtain a new copy of gem5 from Canvas and then rebuild gem5.

Folder Setup: You will be generating a total of 3 different benchmark outputs. However, unlike the first project, you will be modifying parameters from only 1 file, so there is only 1 problem split into various subtasks. In the file "Project 2 Assigned Benchmark" on Canvas, each project group has been assigned a specific benchmark to run. Please carefully read your assigned benchmark. For each of 3 configurations, you will run that benchmark, each of which outputs the files "config.json", "config.ini", and "stats.txt". At the end, you will generate a patch file which is a portable file that makes it easy to apply your gem5 modifications in project 2 to other unmodified gem5 projects. We will use this for grading to ensure that your implementation works. To

generate the described file structure, run the following command as a single command (not 3 separate commands):

```
mkdir Project2-Submission Project2-Submission/Problem1 Project2-
Submission/Problem1/conf1 Project2-Submission/Problem1/conf2 Project2-
Submission/Problem1/conf3
```

Problem1 - Implementing GSelect Branch Predictor

Task 1: Change in BranchPredictor.py

Introduce a new class `GSelectBP()` by adding the following lines to the 'BranchPredictor.py' file, found in the 'gem5/src/cpu/pred' directory.

```
class GSelectBP(BranchPredictor):
      type = 'GSelectBP'
      cxx_class = 'GSelectBP'
      cxx_header = "cpu/pred/gselect.hh"
      PredictorSize = Param.Unsigned(1024, "Size of predictor (entries).")
      PHTCtrBits = Param.Unsigned(2, "Bits per counter.")
      globalHistoryBits = Param.Unsigned(8, "Bits of the global history.")
```

Task 2: Change in SConscript

Add the following lines at the end of the file 'SConscript' in the same directory:

```
                          Source('gselect.cc')
                          DebugFlag('Mispredict')
                          DebugFlag('GSDebug')
```

Task 3: Implement GSelect Functionality in Code

Create two files named gselect.cc and gselect.hh in the directory of `gem5/src/cpu/pred/`. You will need to define and implement the functionality for the five main branch predictor functions: `lookup, update, uncondBranch, btbUpdate, squash`, in addition to any other functions that you need to implement the GSelect branch predictor. Add information about PredictorSize, PHTCtrBits, and globalHistoryBits into the gselect.hh file.

You can look at functions in already implemented branch predictors like 2bit_local and bi_mode, specifically in the 2bit_local.hh and bi_mode.hh files. This will help you understand how to define relevant variables, index counters of the pattern history table for prediction, make updates after the prediction, how to work with the global history, etc.

In your gselect.hh, variable names used for `#ifndef` and `#define` should be different from the bi_mode.hh or the ones used in header files of any other branch predictors.

Once you make the above modifications to fully implement the GSelect branch predictor, rebuild gem5 by running the following command:

```
scons build/ARM/gem5.opt
```

In order to test the accuracy of your implementation, view the "Testing GSelect Implementation" document on Canvas, which walks through the debugging and validation process.

Task 4: Running Benchmarks on your Implementation

Once you are confident that your implementation is correct, you will need to run the benchmarks for the three different configurations just like in project 1. Check the 'Project 2 Assigned Benchmark' spreadsheet on Canvas to determine which benchmark your group has been assigned to run. Then, run only your assigned benchmark for all three configurations of GSelectBP given in the table. For example, if you are assigned "dijkstra_small", then you would run that benchmark on configuration 1, configuration 2, and configuration 3 (after rebuilding after each time you update the file). You don't need to run any other benchmarks. Change the following parameters in 'BranchPredictor.py' to observe 3 different configurations:

Configuration	Predictor size	Counter bits	Global history bits	Benchmark
1	1024	2	8	check "Project 2 Assigned Benchmarks" for your group assignment
2	4096	2	8	
3	4096	2	4	

After modifying 'BranchPredictor.py' for each configuration, make sure to rebuild gem5. You can use `--bp-type` flag to specify the branch predictor type. In this case, we choose GSelectBP.

If you have been assigned the '**dijkstra_small**' benchmark for each configuration, run the following benchmark command (as a single line):

```
./build/ARM/gem5.opt -d [path to target directory]
./configs/example/se.py --cpu-type=DerivO3CPU --caches --l2cache --bp-
type=GSelectBP -c ./benchmarks/dijkstra/dijkstra_small -o
./benchmarks/dijkstra/input.dat
```

If you have been assigned the '**dijkstra_large**' benchmark for each config, run the following benchmark command (as a single line):

```
./build/ARM/gem5.opt -d [path to target directory]
./configs/example/se.py --cpu-type=DerivO3CPU --caches --l2cache --bp-
type=GSelectBP -c ./benchmarks/dijkstra/dijkstra_large -o
./benchmarks/dijkstra/input.dat
```

If you have been assigned the '**qsort_small**' benchmark for each config, run the following benchmark command (as a single line):

```
./build/ARM/gem5.opt -d [path to target directory]
./configs/example/se.py --cpu-type=DerivO3CPU --caches --l2cache --bp-
type=GSelectBP -c ./benchmarks/qsort/qsort_small -o
./benchmarks/qsort/input_small.dat
```

For each configuration, you will run your assigned benchmark and store the results of each in the appropriate directory, which is done by putting the path to the appropriate directory for the configuration as the parameter "path to target directory" in the above commands. You created these directories as the first command in the project instructions.

Creating the Patch File

For this project you will create a patch file, which allows the auto grader to run your implementation of the code on an independent gem5 build. In order to create a correct patch file, please download a new gem5.zip file from the Project1 assignment on Canvas. Unzip the file, change its name to 'gem5_clean' and put it in the same directory as your current gem5 directory. Then run the following command outside the gem5 directory:

```
diff -ruN gem5_clean/src/cpu/pred/ gem5/src/cpu/pred/ > project2.patch
```

NOTE: Make sure the "N" in -ruN is uppercase. You can open project2.patch and locate all your modifications to `SConscript`, `BranchPredictor.py`, `gselect.hh` and `gselect.cc` with a "+" sign at the beginning of each line modified.

Move project2.patch into the Project2-Submission directory. Then you should see project2.patch and the directory Problem1.

Submission Instructions

Submit 3 files to canvas:

1) Submit the file gselect.cc.
2) Submit the file gselect.hh.
3) Submit 'LastName-Project2-Submission.zip. This contains the directory structure outlined in the "Project Deliverables" section. You should be zipping just the folder "Project2-Submission" and then submitting the resulting zip file entitled "LastName-Project2-Submission.zip" where 'LastName' is the last name of the group member submitting the file to Canvas.

Only one group member should make the submission through Canvas. All team members in the group will receive the same score. Please follow the naming conventions correctly so that the auto grader will correctly grade your assignment.

Project Deliverable Structure

1. The project directory structure that you build should contain the following:
    a. A parent directory: Project2-Submission.
    b. 1 file inside Project2-Submission: project2.patch.
    c. 1 directory inside Project2-Submission: Problem1.
    d. Problem1 should contain 3 directories: conf1, conf2, and conf3.
    e. Each of these 3 configuration directories should contain config.ini, config.json, and stats.txt, all generated by gem5 after running the benchmark command of **your assigned benchmark**. The directory structure should be as follows:

```
Project2-Submission
     |
     +---project2.patch
     |
     +---Problem1
         +---conf1
         |   +---config.ini
         |   +---config.json
         |   +---stats.txt
         |
         +---conf2
         |   +---config.ini
         |   +---config.json
         |   +---stats.txt
         |
         +---conf3
             +---config.ini
             +---config.json
             +---stats.txt
```

Rubric

Project 2 [10 points]

a) Accuracy: correct values in the config.ini file and stats.txt file. **[6 points]**

b) Useful comments and clean code: Your implementations of gselect.cc and gselect.hh should contain useful and descriptive comments for each function and variable that you implement. **[2 points]**

c) Working code: The autograder should have no problem in applying the patch and successfully building gem5 with your implementation. **[2 point]**