Homework 5

Due date: Thursday, November 19, 2026, 11:59 PM on Gradescope

Overview:

In HW4 you resolved BEQ, BNE, JAL, and JALR in ID, so every control instruction cost a fetch bubble, even a loop branch that is almost always taken. HW5 predicts in IF and verifies in EX: a correct guess costs nothing, and a wrong predicted next PC flushes the two younger instructions in ID and IF.

Two tables do the guessing: a gshare table predicts whether a conditional branch is taken, and a 64-entry branch target buffer (BTB) predicts where it goes. The ISA, encodings, memory and register semantics, forwarding, load-use stall, and pipeline timing are unchanged from HW4; the HW3/HW4 specs remain the reference for those parts.

What Is Provided:

A skeleton with command-line parsing, the run-loop, the four output writers, the pipeline-register containers, and the memory/register-file constructors. It does not ship with opcode tables, the ALU, sign_extend, or the immediate decoders.

You implement:

- The same six methods as HW3/HW4: **`InsMem.readInstr`**, **`DataMem.readDataMem`**, **`DataMem.writeDataMem`**, **`RegisterFile.readRF`**, **`RegisterFile.writeRF`**, **`FiveStageCore.step`**.
- The four gshare methods: **`GSharePredictor.__init__`**, **`index`**, **`predict`**, **`update`**.
- The five BTB methods: **`BTB.__init__`**, **`index`**, **`tag`**, **`lookup`**, **`update`**.
- The pipeline-register fields your datapath needs; **`State`** starts with only **`nop`** and **`PC`**.

Keep the provided writers, the CLI, `--iodir`, and the four output file names unchanged.

What to Copy Forward:

Most of HW5 is code you already wrote; extend it rather than starting over.

- **From HW3:** the five memory/register-file methods, unchanged (including ignoring writes to x0).

- **From HW3/HW4:** your decoder, opcode tables, ALU, sign-extension/immediate helpers, and HALT detection. They are not supplied and do not change.

- **From HW4:** your `FiveStageCore.step` — State fields, the WB→MEM→EX→ID→IF order, forwarding from EX/MEM and MEM/WB, the load-use stall, HALT drain, and retirement counting. Extend it.

- **New in HW5:** the `GSharePredictor` and BTB classes, prediction fields carried IF→EX, IF prediction, EX resolution and the two-instruction flush, predictor updates, and the new metrics counters.

Setup:

1. **Accept the assignment on Classroom50 here:** https://classroom50.org/cmpe120/cmpe-120-fa26/assignments/hw5/accept, then open the repository in your editor.

2. Follow the README. Inputs, `--iodir`, environment setup, and output names are as in HW4.

3. Keep the writers, CLI, and output names unchanged; the autograder runs `main.py` as before.

Branch Prediction:

Only two things change: IF predicts instead of stalling, and control instructions resolve in EX. Resolving in EX reuses the existing ALU/comparator instead of adding a second ID resolver; the cost is that a miss removes two instructions, not one.

A control instruction has a direction and a target. BEQ/BNE need both; JAL/JALR are always taken, so only the target matters.

gshare: direction

gshare is a table of 2-bit counters, one confidence value per branch-and-history pattern: 0b00 strongly not taken, 0b01 weakly not taken, 0b10 weakly taken, 0b11 strongly taken. Predict taken for 0b10 or 0b11. When a branch resolves, move its counter one step toward the actual outcome (stopping at 0b00 and 0b11) and shift the outcome into the 8-bit global history. Two wrong outcomes in a row are needed to change a confident prediction, which makes it steadier than one bit.

The index XORs the word-aligned PC with the global history: `((PC >> 2) ^ history) & 0xFF`. History lets the same branch use different counters when surrounding patterns differ, so gshare can

learn alternating branches and nested loops. Only BEQ/BNE read and update gshare. Save the index used in IF and carry it to EX: history will have shifted, so recomputing there would update a different counter.

Parameter	Value
Table entries	256
Counter width	2 bits, saturating at 0b00 and 0b11
Initial counter	0b10 (weakly taken)
Prediction	taken when the counter is 0b10 or 0b11
Global history	8 bits, initially 0
Index	((PC >> 2) ^ history) & 0xFF
History update	history = ((history << 1) | taken) & 0xFF

BTB: target

The BTB is a direct-mapped cache of "where this control instruction went last time." Each entry holds a valid bit, a tag, and a 32-bit target. The index selects one entry, the tag proves the entry belongs to this PC, and valid says it has been written.

- Hit: the stored target is a candidate next PC. Miss: no target yet.
- Two instructions can share an index; the tag separates them and the newer entry replaces the older. That is a conflict miss, not a bug.

- Install or refresh the entry when a control instruction resolves taken; do not install one for a not-taken conditional branch.

The index and tag fields are fixed for this assignment:

Parameter	Value
Entries	64, direct-mapped
Index	(PC >> 2) & 0x3F
Tag	(PC >> 8) & 0xFFFFFF
Entry contents	valid bit, tag, 32-bit target
Lookup	hit only when valid is set and the tag matches
Update	install the target when a control instruction resolves taken

Predicting in IF

For BEQ/BNE ask gshare; for JAL/JALR the direction is taken; all four look up the BTB. The predicted next PC is the BTB target when the prediction is taken and the lookup hits, otherwise PC + 4. Carry the predicted PC, direction, and gshare index to EX. A BTB miss on a taken prediction fetches PC + 4: a missing target, not necessarily a wrong path.

Resolving in EX

Compute the actual next PC from forwarded operands: BEQ/BNE = PC + B-immediate if taken, otherwise PC + 4; JAL = PC + J-immediate; JALR = (rs1 + I-immediate) & ~1. Compare it with the predicted PC:

- Equal: no bubble.
- Different: keep the control instruction, flush ID and IF, and redirect to the actual PC. That is the two-cycle misprediction penalty.

Update gshare for BEQ/BNE (counter, then history) and the BTB for any taken control instruction.

Subtle case: predicted taken + BTB miss fetches PC + 4. If the branch is actually not taken, the path was right even though the direction was wrong. Update the counter but do not flush. Flushing compares next PCs, not directions.

Jumps, load-use, and HALT

- JAL/JALR are always predicted taken; a BTB miss or wrong target gets the same two-instruction flush.
- A control instruction using a load's result stalls in ID like any load-use consumer; EX resolves after the value is available.
- HALT stops fetch and is not retired. Discard a HALT fetched on a flushed path.

A good order to build it

1. Resolve control in EX with no prediction (always PC + 4); confirm the flush and that RF/DMEM still match.
2. Add the BTB target; check test 14.
3. Add gshare direction; check test 15.
4. Add the counters and metrics lines; compare with the expected file.

Cycles:

A correct prediction adds no bubble. A misprediction costs two cycles (ID and IF become bubbles). The first taken use of a branch also flushes because the BTB is cold, even when the direction was right. HW4 paid one bubble for every control instruction; HW5 pays zero when right and two when wrong or when

the BTB has no target. HW5 therefore wins on long loops and can lose on short straight-line control sequences. Retirement, HALT, and bubble rules are unchanged.

Output Files:

RFResult, DMEMResult, and TraceResult are unchanged. PerformanceMetrics keeps the four HW3/HW4 lines and adds:

Line	Meaning
Conditional Branches	BEQ/BNE resolved in EX
Branch Mispredictions	conditional branches whose predicted direction differed from actual
Branch Prediction Accuracy	100*(branches - mispredictions)/branches; 100 when there are none
BTB Lookups	resolved control instructions (BEQ/BNE/JAL/JALR)
BTB Hits	lookups with a valid, tag-matching entry at fetch
BTB Hit Rate	100*hits/lookups; 100 when there are no lookups
Control Mispredictions	control instructions whose predicted next PC differed from actual (these flush two)

Trace files keep one row per cycle, -1 for a bubble. A wrong-path instruction may appear in IF/ID for a cycle but must never reach WB.

Example:

This is `tests/15_branch_backward_loop`: a counted loop whose back edge is taken four times and then not taken once.

```
0   ADDI x1, x0, #0

4   ADDI x2, x0, #5

8   ADD  x1, x1, x1

12  ADDI x2, x2, #-1

16  BNE  x2, x0, #-8      ; back to PC 8 while x2 != 0

20  SW   x1, 0(x0)

24  HALT
```

The BNE at PC 16 is the only control instruction:

Fetch	gshare says	BTB	predicted → actual	result
1st	taken (0b10)	miss	20 → 8	flush 2; install target 8
2nd	taken (0b11)	hit	8 → 8	no bubble
3rd	taken (0b11)	hit	8 → 8	no bubble
4th	taken (0b11)	hit	8 → 8	no bubble

5th	taken (0b11)	hit	8 → 20	flush 2

First fetch: direction hit, target miss, predicted 20, actual 8, so two instructions flush; the BTB installs target 8 and the counter moves to 0b11. The next three predict 8 and get 8, so no bubbles. The fifth predicts 8 but the loop exits at 20, so two more flush.

Result: 4/5 directions correct (80%), 4/5 BTB hits (80%), 2 control mispredictions. The expected metrics file is:

```
Total Cycles: 26

Instructions Executed: 18

CPI: 1.444444

IPC: 0.692308

Conditional Branches: 5

Branch Mispredictions: 1

Branch Prediction Accuracy: 80.000000

BTB Lookups: 5

BTB Hits: 4

BTB Hit Rate: 80.000000

Control Mispredictions: 2
```

If your trace diverges, find the first cycle where it differs. A missing first-pass flush usually means the BTB was updated too early; a missing final-pass flush usually means the predicted and actual next PCs were not compared.

Running & Testing:

Run and test as in HW4:

```
python main.py --iodir tests/15_branch_backward_loop/input

python run_tests.py

python run_tests.py 14 15
```

Add `--verbose` to print one line per clock and watch a prediction fail. You can still write programs in `asm/` and build them with `assemble.py`.

Gradescope Submission:

25 cases, 8 public and 17 hidden, each with four output files worth 1 point: 4 points per case, 100 total. A case scores only when all four files match.

RFResult and DMEMResult must match exactly. TraceResult must retire the same PC sequence (bubble positions may differ). In PerformanceMetrics, Instructions Executed, Conditional Branches, and BTB Lookups must be exact; Total Cycles, CPI, and IPC allow 2% error; Branch Prediction Accuracy and BTB Hit Rate allow 5 percentage points (or 5% relative, whichever is larger).

Submission mechanics are identical to HW4: keep `main.py`, do not change `--iodir` or the output names, do not modify the writers, stay within the 10-second-per-case limit, and submit through GitHub on Gradescope.

Troubleshooting:

For decoder, byte-order, sign-extension, HALT, forwarding, and load-use issues, see the HW3/HW4 lists; those do not change.

- `NotImplementedError` in a predictor class: the predictor is still empty.
- Accuracy near 50% or a very low BTB hit rate: check the history shift, the index, the tag, the valid bit, and that the BTB updates only on taken.

- A correct prediction still costs a bubble: compare the predicted and actual next PCs, not the directions, and check the fields you carried to EX.
- A wrong-path instruction retires: EX must flush both ID and IF and keep the control instruction itself.
- A direction miss on the correct fall-through path flushes unnecessarily: after a BTB miss, a taken prediction fetched PC + 4, so an actually-not-taken branch has no wrong path to flush.
- Stops at a speculative HALT: cancel it when a control instruction flushes the younger instructions.
- CPI far from the reference: verify HW4 forwarding and load-use, that a correct prediction adds no bubble, and that a BTB hit uses its target.