ECS 154B Lab 5, Spring 2015 (Optional)
Due by 11:55 PM on June 5, 2015
Via SmartSite

Objectives

- Implement a 2-2 Gshare branch predictor.

Description

As you have learned during class, one of the biggest slow downs in pipelined CPUs are control hazards. The best way to overcome them is through prediction. In the last lab, you used a static branch not taken prediction strategy for your pipelined CPU, but now you will be designing a dynamic Gshare branch predictor module.

Gshare

The Gshare branch predictor has 2 parameters: **m** and **n**.

- **m**: the amount of states to keep.
  - This determines the size of the branch history register, the number of PC bits to use, and the number of saturating up/down counters.
- **n**: the number of bits in each saturating up/down counter.

Branch History Register

An **m** bit long shift register that keeps track of whether the last **m** branches were taken or not.

- For example, if **m** is 2, and the branch history register is 11, it means the last two branch instructions were taken. If the next branch is not taken, then the branch history register will contain 10. If the branch after that is taken, then the branch history register = 01. If the next branch is taken then the branch history register = 11.
- Only branch instructions change the contents of the branch history register.

Counters

There are 2^m counters, numbered from 0 to $2^m - 1$. Each counter is an **n** bit saturating counter. Saturating means that when the counter is at its maximum value and told to count up, it stays at the maximum value. When it is at 0 and told to count down, it stays at 0. For example, if **n** is 2, the counter's current value is 3, and count up is high then the value of the counter will stay at 3. Likewise, if the current value of the counter is 0 and count down is high, then the value of the counter remains at 0.

If **n** is equal to 2, one way to interpret each potential value in the counter is as follows:

- 00: Strongly not taken.
- 01: Weakly not taken.
- 10: Weakly taken.
- 11: Strongly taken.

Tag Registers

There are 2^m tag registers, numbered from 0 to $2^m - 1$. Each register stores the value of the PC associated with the counter value of the same number.

Branch Destination Registers

There are 2^m tag registers, numbered from 0 to $2^m - 1$. Each register stores the destination of the branch, and is the size of the PC. For example, if the PC is 32 bits long, then each branch destination register is 32 bits wide.

Prediction

Gshare makes predictions on a PC address in the following way:

- It takes the bottom **m** bits of the PC and `XOR`s them with the branch history register to form an **Index**.
- Gshare will predict true if Counters[Index] $\geq 2^{n-1}$ && TagRegisters[Index] == PC.
- If not, Gshare will predict false.

The branch destination address of the branch instruction is BranchDestinationRegisters[Index].

- In the above expression PC{sizeof(PC)...m} are all the bits in the PC except for the bottom **m**.
- Also, note that it is **n** and not **m** in Counters[Index] $\geq 2^{n-1}$.

Update

Gshare is updated after we figure out if the instruction is a branch, and if it was actually taken or not.

Instruction = Branch

XOR the bottom **m** bits of the PC with the branch history register to form an **Index**.

```
if(TagRegister[Index] == PC && the branch was actually taken)
   Counter[Index] = Counter[Index] + 1
else if(TagRegister[Index] == PC && the branch was actually not taken)
   Counter[Index] = Counter[Index] - 1
else if(TagRegister[Index] != PC && the branch was actually taken)
   Counter[Index] = 1
else if(TagRegister[Index] != PC && the branch was actually not taken)
   Counter[Index] = 0

TagRegisters[Index] = PC
BranchDestinationRegisters[Index] = Branch Destination of Instruction
Shift 1 into the branch history register if the branch was taken
Shift 0 into the branch history register if the branch was not taken
```

Instruction $\neq$ Branch

In this case, Gshare is not updated.

Details

You are to implement a 2-2 Gshare branch predictor that works as outlined above. You are allowed to use all of the modules and features of Logisim.

Inputs

Input Signals		
Input Name	Bit Width	Description
PredPC	32	The instruction address that we would like to make a prediction on.
UpdatePC	32	The instruction address that we are using to update the Gshare predictor.
IsBranch	1	Is the instruction associated with **UpdatePC** a branch instruction? 1 if true, 0 if false.
BranchTaken	1	Was the branch instruction associated with **UpdatePC** actually taken? 1 if true, 0 if false.
BranchDestAddr	32	The branch destination of the branch instruction associated with **UpdatePC**.

Outputs

Output Signals		
Output Name	Bit Width	Description
TakeBranch	1	Your prediction on whether the instruction at **PredPC** should be taken or not. 1 if true, 0 if false.
PredBranchDestAddr	32	The branch destination address associated with the instruction at **PredPC**.

Grading

You will be provided with two circuits for testing.

InputGen

Inputs

- **Clock**: The system clock.

Outputs

- The input signals to your Gshare predictor, as specified in the Input Signals table.

Test

Checks whether the output of your predictor is correct.

Inputs

- **TakeBranch**: Your prediction on whether the instruction at **PredPC** should be taken or not.
- **PredBracnhAddr**: The branch destination address associated with the instruction at **PredPC**.
- **Clock**: The system clock.

Outputs

- **IsPredCorrect**: Was your prediction correct? 1 if true, 0 if false.
- **CorrectPred**: The prediction that should be made.
- **IsPredBranchCorrect**: Was the branch address associated with your prediction correct? 1 if true, 0 if false. If the prediction is supposed to be 0, this will always be 1.
- **CorrectPredBranch**: The correct branch address associated with **PredPC**. Ignore if **CorrectPred** is 0.
- **Correct**: Are both **IsPredCorrect** and **IsPredBranchCorrect** correct? 1 if true, 0 if false.
- **Done**: Is the test finished? 1 if true, 0 if false. There are 28 instructions, so this will be 1 once the clock has gone high 28 times.

A 28 entry shift register is connected to the output of correct. Once the test is over, it should be completely filled with ones if your circuit is functioning correctly. You will also be given a Python file that contains a class which implements an **m-n** Gshare branch predictor. You can work with the file to help you better understand how a Gshare branch predictor works.

Rubric

Implementation

- Weight: 100%
- Points: 28
  - Grading Scheme: 1 point for each 1 in the 28 bit output shift register.

This assignment is optional. If your grade on Lab 5 is higher than any of your grades on Labs 1 through 3, the score you receive will replace the lowest of those grades. If you scored 100% on all three labs, enjoy your break. There will be no interactive grading for this assignment.

Submission

Submit your circuit on SmartSite, along with a README file. In the README, include:

- Name of Partner 1, student ID number
- Name of Partner 2, student ID number
- Whether or not your circuit works, and if it does not, what it does not work on.
- Any difficulties you had.
- Anything else you feel the TAs should know.

Hints

- Use copy and paste to make your registers and counters, rather than doing it by hand.