This Set

Practice Problems

Exams and assignments below available at `https://www.ece.lsu.edu/ee4720/prev.html`.

Use problems below to practice material in this set.

Some solutions are detailed and are useful for understanding material.

Analysis Problems

2020 fep3: Local history, PHT interference.

2017 fep3a: TNTTnnn TnTTtttt bimodal, var pattern len. local. Hist sz. GHR.

2016 fep3a: B2: TNTNTNT N (nn or tt) bimodal. local. min LH

2013 fep3: B1: TNTTTN, B2: TNTrTN, B3: T.. bimodal, local. PHT colli. GHR

2014 fep3: TTTNN, B2: rrrqqq (grps of 3) bimodal. local GHR val

2015 fep3: NTTNNN, B2: T2,4,6NNNN, B3: T.. bimodal, local, min GHR siz

Branch Predictor Variations, and Hardware

2019 Final Exam Problem 3b: Update gshare GHR using predicted outcome.

2018 Final Exam Problem 4b: Convert local predictor to global predictor.

2016 fep3 (b) Post-loop branch on global predictor variations.

2017 fep3 (b): Convert illustrated bimodal into local predictor.

2013 fep3b: Draw a digram of local predictor.

CTI (Control Transfer Instruction):

Any instruction that causes instruction fetch to switch to another location, the target, (either immediately or after the execution of a delay-slot instruction). This includes branches, jumps, calls, and traps.

Direction and Target Prediction

Branch Direction Prediction:

Prediction of the outcome of branch. (Whether taken or not taken.)

There are many methods of predicting branches.

One estimate is $10P$ different predictors …
… where P is the number of computer engineering professors.

Fortunately the most important ones are use a few simple techniques.

CTI Target Prediction:

Prediction of the target of a branch or of other CTI's.

CTI Prediction Motivation

The 5-stage MIPS scalar pipeline executes CTIs without penalty.

Penalty cannot be avoided in other implementations ...
... such as 2-way superscalar ...
... or a 10-stage pipeline.

Branches occur frequently in code, about one in six in some integer code.

Without prediction their impact on performance will be significant.

For example, code can take 75% longer on a 4-way superscalar pipeline.

Example: Estimate of impact on 4-way superscalar 5-stage MIPS:

Assume that one out of six instructions is a branch ...
... and that the code is perfectly scheduled so that there are no stalls.

Ideal time to execute N instructions: $\lceil N/4 \rceil$.

Number of squashed instructions if branch is in ...
... Slot 0: $2+4$, Slot 1: $1+4$, Slot 2: $0+4$, Slot 3: $0+3$. ...
... Average: 4.5.

Number of squashed instructions: $N\frac{1}{6}4.5 = 0.75N$.

Execution time with squashed instructions: $(1+0.75)N/4$.

Slowdown due to squashes: $\frac{(1.75)N/4}{N/4} = 1.75$.

That's 75% longer. *Seventy-five percent!*.

Methods Covered

Simple: the *Bimodal Predictor*

A.k.a. the *One-level predictor*

Commonly used in simpler CPUs.

Correlating (Two-Level) Predictors

Local History, a.k.a. *PAg*.

Global History, a.k.a. *GAg*.

gshare.

Commonly used in general purpose CPUs.

Branch Prediction Idea

Idea: Predict using past behavior.

Typical Behaviors

```
A: beq r2, r0, ERROR   # N N N N N N N N ...         # Never (or very rarely) taken.

B: bne r3, r0, LOOP    # T T N   T T N   T T N ...   # Looks like a 3-iteration loop.

C: bltz r4, SKIP       # T T N N  T T N N  T T N N ... # Arbitrary repeating pattern.

D: bc1t LINEX          # .. T T T T T N N N N N ...  # Long runs of Ns and Ts.

   lw    r1, 0(r2)     # Random number: 0 or 1.
E: beq   r1, r0 SKIP   # T N N T   N T T T N   ...   # Random, no pattern.
```

Terminology and Execution Example

Example: Default (for EE 4720) execution timing on a 2-way superscalar MIPS implementation with branch prediction.

In cycle 0, when the branch is in `IF`, it is *predicted* taken.

In cycle 1, when the branch is in `ID`, the predicted target, `TARG`, is fetched.

Alas, the branch has been *mispredicted*, this is discovered when the branch is *resolved*, in `EX` by the ALU near the end of cycle 2.

Instructions from `lw` to `jr` are *speculatively executed* from cycle 1 to 3.

In cycle 3, when the branch is in `ME`, the *wrong-path* instructions, `lw`, `addi`, ..., `jr`, are squashed, part of *misprediction recovery*.

In cycle 4 the *correct-path* instructions are finally fetched, starting at the branch *fall-through* instruction, `sub`.

The *misprediction penalty* in this example is 3 cycles.

```
 # Cycle             0  1  2  3  4  5  6  7  8
 beq r1, r2, TARG    IF ID EX ME WB
 add r2, r3, r4      IF ID EX ME WB
 sub r6, r7, r8                  IF ID EX ME WB
 or r9, r10, r11                 IF ID EX ME WB

TARG:
 lw r7, 4(r6)           IF ID EXx
 addi r6, r6, 8         IF ID EXx
 xori r7, r7, 0xaa         IF IDx
 andi r9, r7, 0xff         IF IDx
 sw r20, -4(r6)               IFx
 jr r31                       IFx
 # Cycle             0  1  2  3  4  5  6  7  8
```

Branch Prediction Terminology

Outcome: [of a branch instruction execution].
Whether the branch is taken or not taken.

Fall-Through Instruction: [of a branch]
For ISAs without branch delay slots, the instruction after the branch; for ISAs with a delay slot, the instruction after the branch's delay slot instruction.

T:
A taken branch. Used in diagrams to show branch outcomes.

N:
A branch that is not taken. Used in diagrams to show branch outcomes.

Prediction:
A direction determined by a branch direction predictor or a target determined by a target predictor.

Resolve: [a branch].
To determine whether a branch is taken and if so, to which address.

In our default MIPS **without** prediction branches resolved in `ID`.

In our default MIPS with prediction branches resolved at the end of `EX`.

Misprediction:
An incorrect prediction.

Wrong-Path Instructions:
Instructions fetched due to a misprediction. These instructions can start at the target (when the branch is predicted taken) or at the fall-through when predicted not taken.

Wrong-path instructions must not be allowed to finish.

Correct-Path Instructions:
Instructions fetched based on the correct outcome of the branch, starting either at the target or fall-through.

Prediction Accuracy: [of a prediction scheme].
The number of correct predictions divided by the number of predictions.

Speculative Execution:

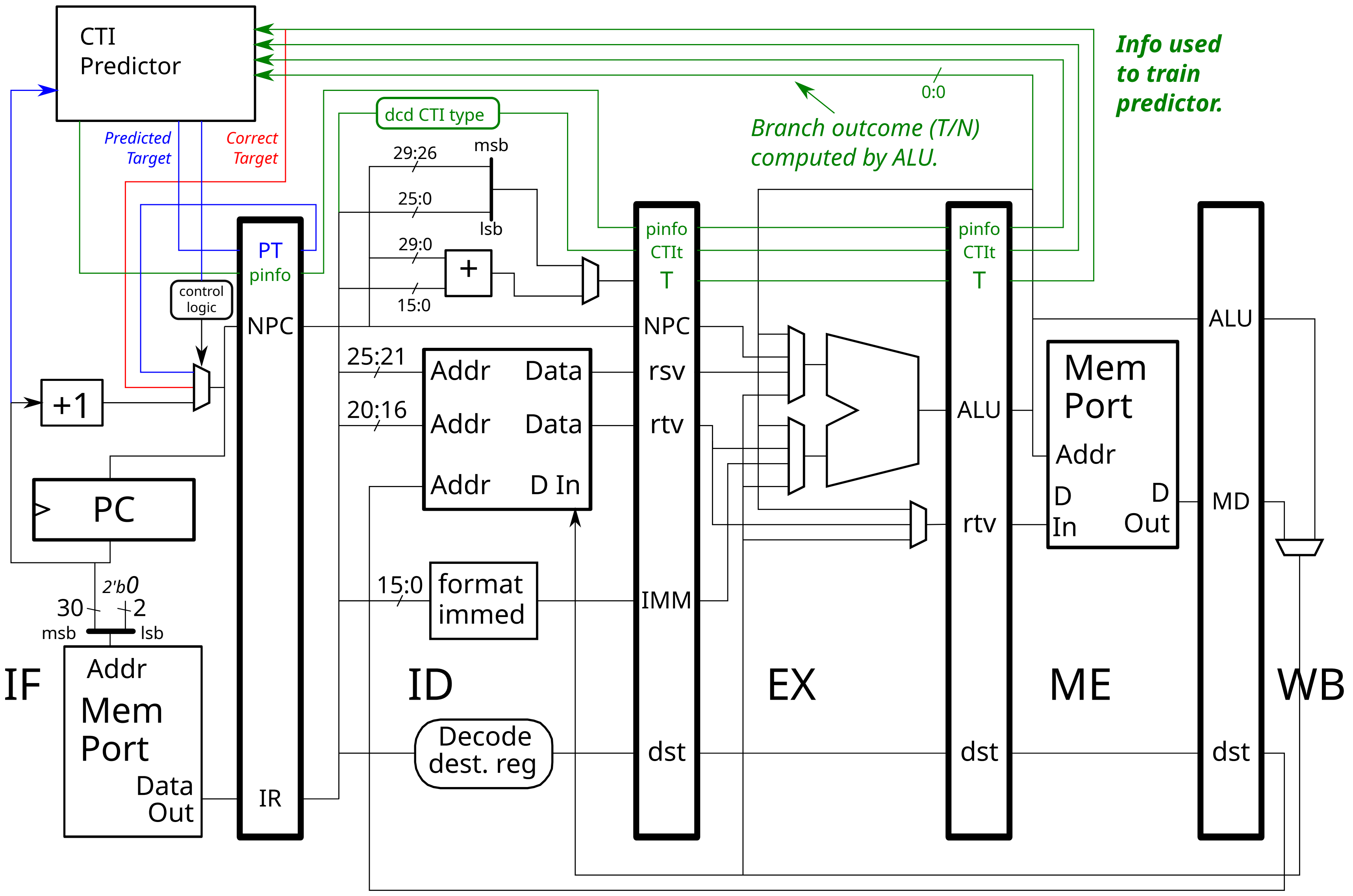
The execution of instructions which may not be on the correct program path (due to a predicted CTI) or which may not be correct for other reasons (such as load/store dependence prediction [a topic that is usually not covered in this class]).

Misprediction Recovery:

Undoing the effect of speculatively executed instructions ...

... and re-starting instruction fetch at the correct address.

Hardware Overview




```
# Cycle              0  1  2  3  4  5  6  7
beq r1, r2, TARG     IF ID EX ME WB
add r2, r3, r4       IF ID EX ME WB
sub r6, r7, r8                   IF ID EX ME
or r9, r10, r11                  IF ID EX ME

TARG:
 lw r7, 4(r6)           IF ID EXx
 addi r6, r6, 8         IF ID EXx
 xori r7, r7, 0xaa         IF IDx
 andi r9, r7, 0xff         IF IDx
 sw r20, -4(r6)               IFx
 jr r31                       IFx
# Cycle              0  1  2  3  4  5  6  7
```

Bimodal Branch Predictor

Bimodal Branch Predictor:

A branch direction predictor that associates a 2-bit counter (just a 2-bit unsigned integer) with each branch. The counter is incremented when the branch is taken and decremented when the branch is not taken. The branch is predicted taken if the counter value is 2 or 3.

Example of 2-Bit Counter Used for Four-Iteration Loop

In diagram below initial counter value assumed to be zero.

```
 # Counter:        0  1  2  3  2   3  3  3  2   3  3  3  2   3  3  3  2
 beq r1, r2, TARG    T  T  T  N   T  T  T  N   T  T  T  N   T  T  T  N
 # Prediction       n  n  t  t   t  t  t  t   t  t  t  t   t  t  t  t
 # Outcome:          x  x     x            x            x            x
```

Prediction Accuracy: $\frac{3}{4}$, based on repeating pattern.

Bimodal Branch Predictor

Characteristics:

Low cost.

Used in many 20th century processors.

Bimodal Branch Predictor Idea

Idea: maintain a *branch history* for each branch instruction.

Branch History:
Information about past behavior of the branch.

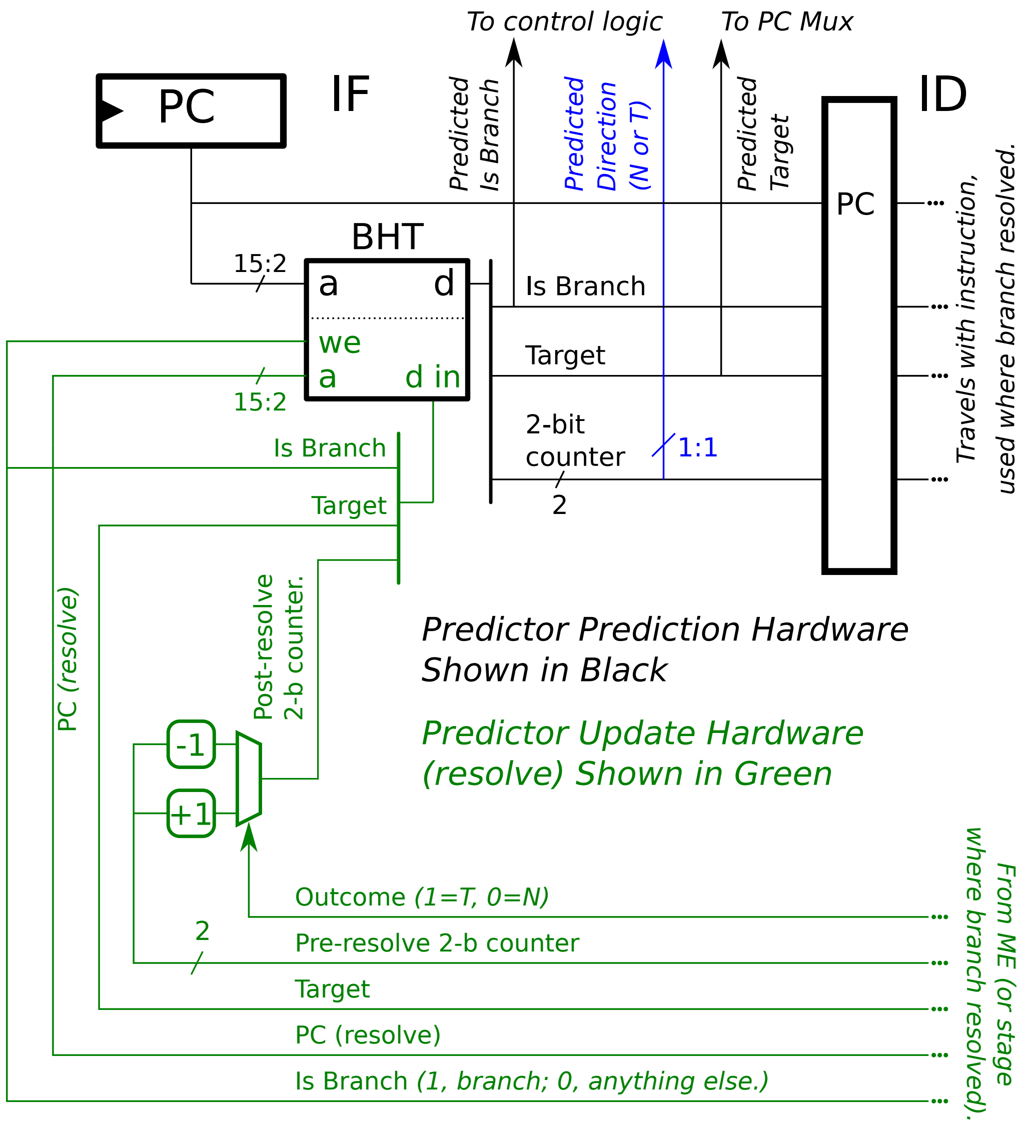
Branch histories stored in a *branch history table (BHT)*.

Often, branch history is sort of number of times branch taken. . .
. . . minus number of times not taken.

Other types of history possible.

Branch history read to make a prediction.

Branch history updated when branch outcome known.



Branch History Counter

Branch History Counter and *Two-Bit Counter*

If a counter used, branch history incremented when branch taken. . .
. . . and decremented when branch not taken.

Symbol n denotes number of bits for branch history.

To save space and for performance reasons . . .
. . . branch history limited to a few bits, usually $n = 2$.

Branch history updated using a *saturating counter*.

A saturating counter is an arithmetic unit that can add or subtract one . . .
. . . in which $x + 1 \rightarrow x + 1$ for $x \in [0, 2^n - 2]$ . . .
. . . $x - 1 \rightarrow x - 1$ for $x \in [1, 2^n - 1]$ . . .
. . . $(2^n - 1) + 1 \rightarrow 2^n - 1$ . . .
. . . and $0 - 1 \rightarrow 0$.

For an n-bit counter, predict taken if counter $\geq 2^{n-1}$.

Branch History (2-Bit) Counter Example

Example of 2-Bit Counter Used for Four-Iteration Loop

In diagram below initial counter value assumed to be zero.

```
# Counter:        0  1  2  3  2   3  3  3  2   3  3  3  2   3  3  3  2
beq r1, r2, TARG    T  T  T  N   T  T  T  N   T  T  T  N   T  T  T  N
# Prediction       n  n  t  t   t  t  t  t   t  t  t  t   t  t  t  t
# Outcome:          x  x     x            x            x            x
```

Prediction Accuracy: $\frac{3}{4}$, based on repeating pattern.

Predictor Hardware

Bimodal aka *One-Level Branch Predictor* Hardware

Illustrated for 5-stage MIPS implementation.

Branch Prediction Steps

1: *Predict*.

Read branch history, available in IF.

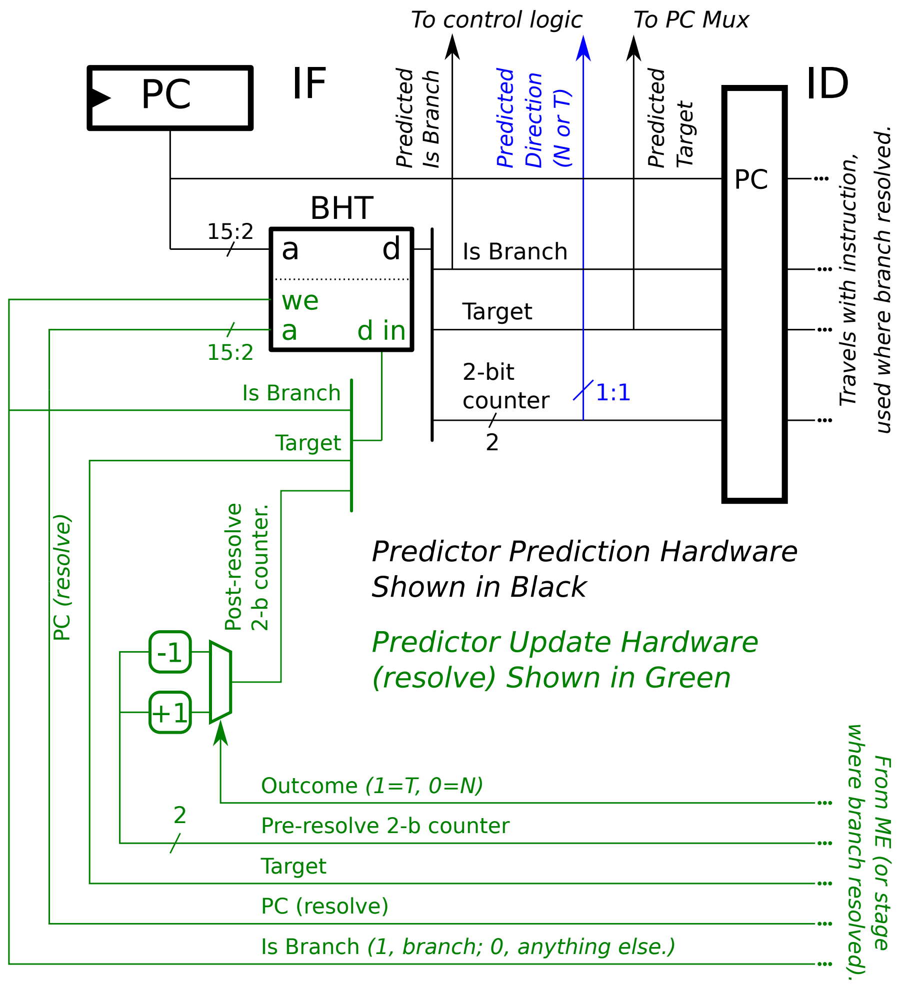
2: *Resolve* (Determine Branch Outcome)

Execute predicted branch in usual way.

3: *Recover* (If necessary.)

Undo effect of speculatively executing instructions, start fetching from correct path.

4: *Update* Branch History



Branch History Table

Branch History Table

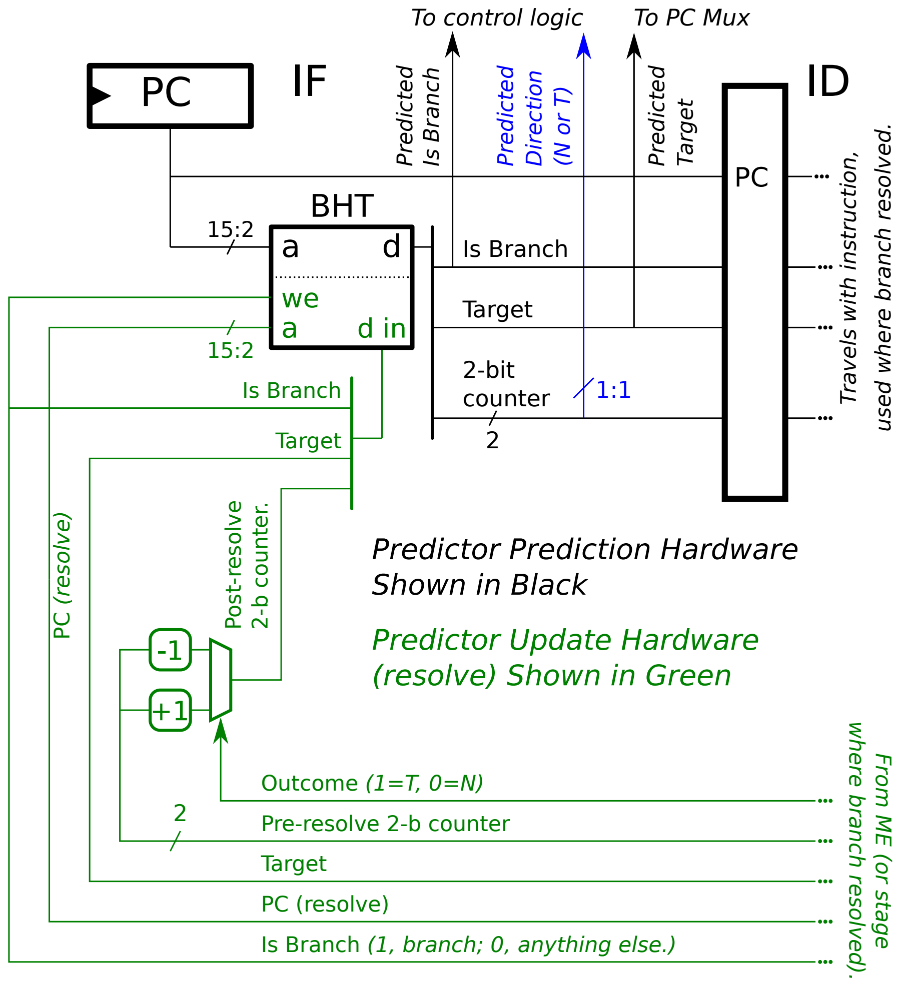
Stores info about each branch.

Used in all branch predictors, the info varies based on predictor type.

Implemented using a memory device.

Address (called index) is hash of branch address (PC).

For 2^m-entry BHT, hash is m lowest bits of branch PC **skipping alignment**.



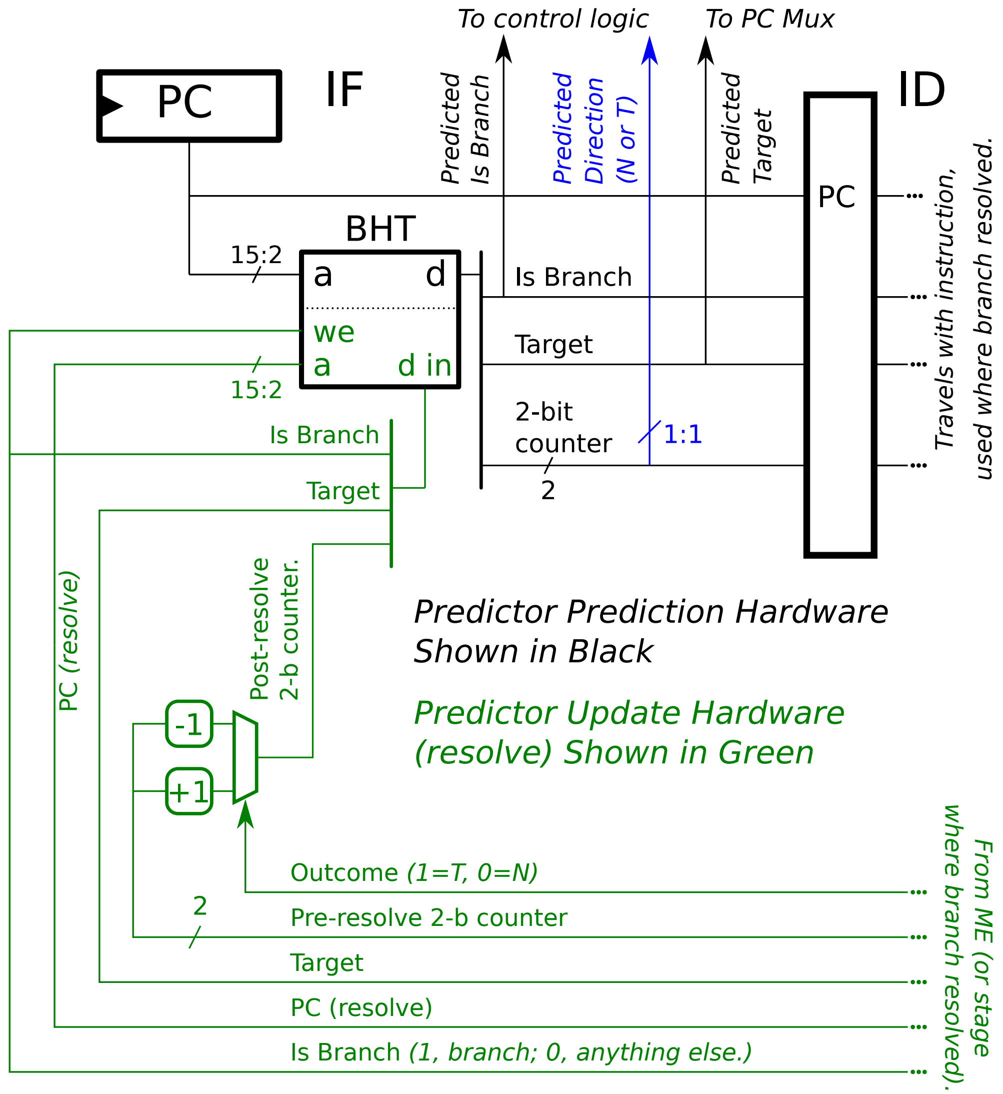
Output of BHT

CTI Type, indicating whether insn is a branch, jump, etc.

Note: *CTI*, Control Transfer Instruction, is any instruction that causes execution to go somewhere else, such as a branch, jump, or trap.

Target Address, the address to go to if CTI taken.

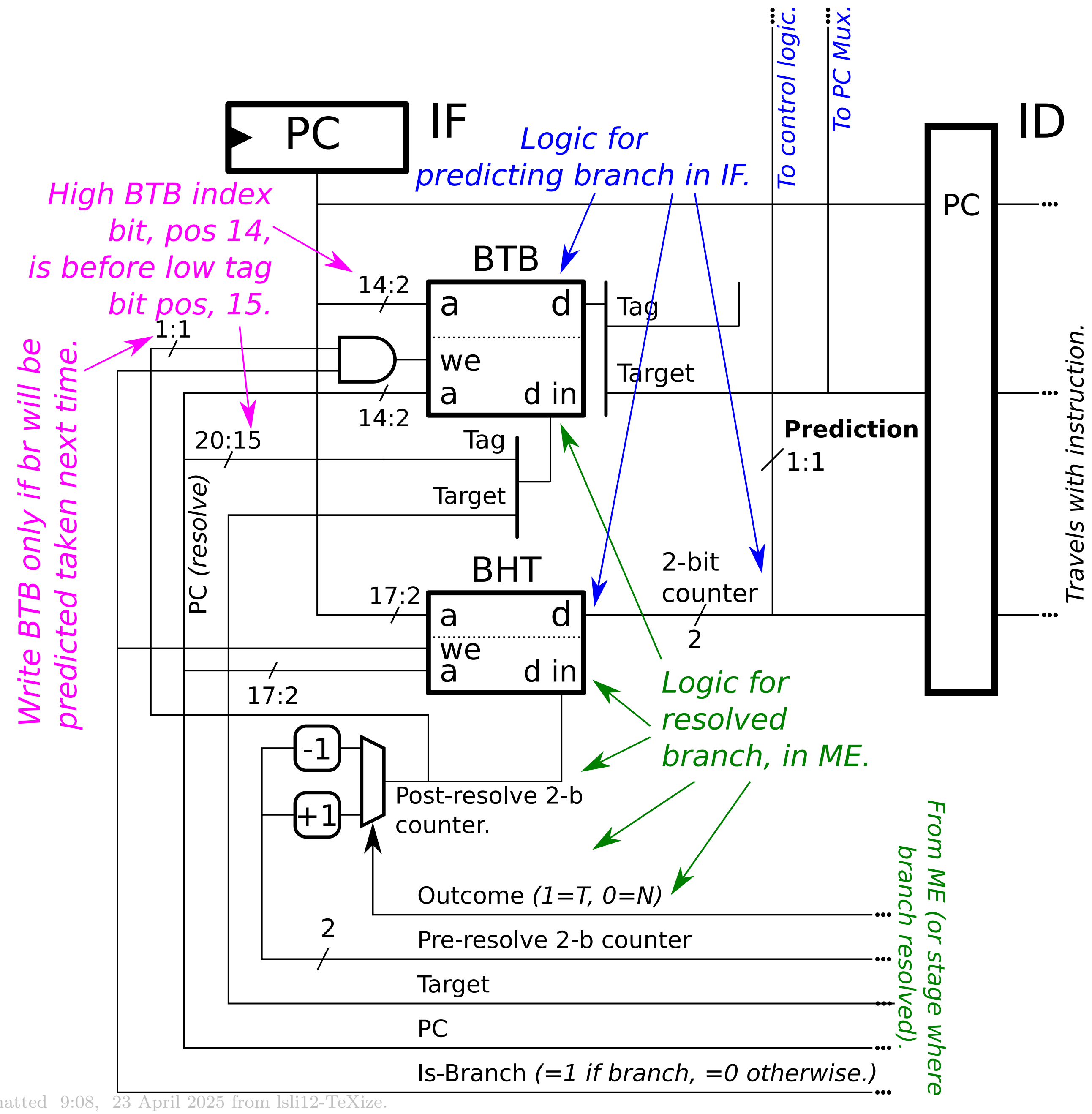
Two-Bit Counter, bias in taken direction.



Separate Target and History Tables

Use separate tables for 2-bit counter (BHT) and other info (BTB).

Use a tag to detect some collisions.



Sample Local Histories

Outcomes for individual branches, categorized by pattern, sorted by frequency.

Branches running TEX text formatter compiled for SPARC (Solaris).

```
Arbitrary, pat 60288, br732164, 0.7743 0.7170 0.7199  (0.19675)
          % Patterns # Branches gshre local  corr   Local History
  0:     fe7f  0.0004     1397  0.912 0.916 0.896  TTTTTTTNNTTTTTTT  0
  1:     ff3f  0.0004     1323  0.924 0.909 0.900  TTTTTTNNTTTTTTTT  0
  2:     fcff  0.0004     1317  0.949 0.939 0.948  TTTTTTTTNNTTTTTT  0
  3:     ff9f  0.0003     1245  0.910 0.905 0.898  TTTTTNNTTTTTTTTT  0
  4:     f9ff  0.0003     1235  0.955 0.950 0.955  TTTTTTTTTNNTTTTT  0
  5:     ffcf  0.0003     1188  0.926 0.921 0.923  TTTTNNTTTTTTTTTT  0
  6:       60  0.0003     1163  0.873 0.829 0.854  NNNNNTTNNNNNNNNN  0
  7:      180  0.0003     1159  0.955 0.914 0.926  NNNNNNNTTNNNNNNN  0
  8:      300  0.0003     1158  0.949 0.926 0.934  NNNNNNNNTTNNNNNN  0
  9:       c0  0.0003     1155  0.944 0.917 0.926  NNNNNNTTNNNNNNNN  0
```

```
Short Loop, pat 124, br  137681, 0.8908 0.9055 0.7441  (0.03700)
          % Patterns # Branches gshre local  corr   Local History
  0:     5555  0.0040    14753  0.987 0.981 0.912  TNTNTNTNTNTNTNTN  1
  1:     aaaa  0.0040    14730  0.859 0.978 0.461  NTNTNTNTNTNTNTNT  1
  2:     9249  0.0022     8062  0.997 0.992 0.988  TNNTNNTNNTNNTNNT  1
  3:     4924  0.0022     8055  0.997 0.998 0.998  NNTNNTNNTNNTNNTN  1
  4:     2492  0.0022     8047  0.993 0.991 0.009  NTNNTNNTNNTNNTNN  1
  5:     db6d  0.0013     4864  0.713 0.915 0.065  TNTTNTTNTTNTTNTT  1
  6:     b6db  0.0013     4713  0.862 0.903 0.926  TTNTTNTTNTTNTTNT  1
  7:     6db6  0.0012     4640  0.991 0.978 0.970  NTTNTTNTTNTTNTTN  1
  8:     bbbb  0.0008     3061  0.896 0.936 0.949  TTNTTTNTTTNTTTNT  1

Long Loop?, pat 32, br  185795, 0.9170 0.9052 0.9096  (0.04993)
  0:     fffe  0.0025     9204  0.902 0.930 0.913  NTTTTTTTTTTTTTTT  2
  1:     8000  0.0025     9198  0.654 0.700 0.705  NNNNNNNNNNNNNNNT  2
  2:     7fff  0.0022     8052  0.890 0.817 0.818  TTTTTTTTTTTTTTTN  2
  3:     ffbf  0.0018     6800  0.933 0.908 0.920  TTTTTTNTTTTTTTTT  2
  4:     feff  0.0018     6782  0.946 0.938 0.942  TTTTTTTTNTTTTTTT  2
  5:     ff7f  0.0018     6778  0.949 0.946 0.950  TTTTTTTNTTTTTTTT  2
  6:     fdff  0.0018     6738  0.947 0.941 0.946  TTTTTTTTTNTTTTTT  2
  7:        1  0.0018     6690  0.955 0.945 0.942  TNNNNNNNNNNNNNNN  2
  8:     fffd  0.0018     6667  0.968 0.966 0.967  TNTTTTTTTTTTTTTT  2
```

```
Phase Change, pat 26, br 48190, 0.8453 0.9040 0.8470  (0.01295)
          % Patterns # Branches gshre local  corr   Local History
  0:     c000  0.0012     4554  0.653 0.777 0.680  NNNNNNNNNNNNNNTT  3
  1:     e000  0.0009     3420  0.714 0.859 0.758  NNNNNNNNNNNNNTTT  3
  2:     f000  0.0008     2942  0.756 0.888 0.788  NNNNNNNNNNNNTTTT  3
  3:     fffc  0.0008     2878  0.908 0.960 0.959  NNTTTTTTTTTTTTTT  3
  4:     f800  0.0007     2642  0.786 0.917 0.827  NNNNNNNNNNNTTTTT  3
  5:        3  0.0007     2572  0.968 0.952 0.951  TTNNNNNNNNNNNNNN  3
  6:     fc00  0.0007     2435  0.815 0.933 0.854  NNNNNNNNNNTTTTTT  3
  7:     fe00  0.0006     2225  0.836 0.936 0.876  NNNNNNNNNTTTTTTT  3
  8:     ff00  0.0006     2140  0.856 0.947 0.931  NNNNNNNNTTTTTTTT  3
  9:     ff80  0.0006     2061  0.854 0.941 0.934  NNNNNNNTTTTTTTTT  3


One Way, pat    2, br  2617433, 0.9917 0.9934 0.9897  (0.70337)
  0:     ffff  0.5151  1916950  0.993 0.996 0.993  TTTTTTTTTTTTTTTT  4
  1:        0  0.1882   700483  0.988 0.986 0.982  NNNNNNNNNNNNNNNN  4
```

Two-Level Correlating Predictors

Idea: Base branch decision on . . .

. . . the address of the branch instruction (as in the one-level scheme) . . .

. . . and the most recent branch outcomes.

History:

The outcome (taken or not taken) of the most recent branches. Usually stored as a bit vector with 1 indicating taken.

Pattern History Table (PHT):

Memory for 2-bit counters, indexed (addressed) by some combination of history and the branch instruction address.

Some Types of Two-Level Predictors

Global, a.k.a. *GAg*.

History is global (same for all branches), stored in a *global history register* (*GHR*).

PHT indexed using history only.

gshare

History is global (same for all branches), stored in a *global history register* (*GHR*).

PHT indexed using history exclusive-ored with branch address.

gselect

History is global (same for all branches), stored in a *global history register* (*GHR*).

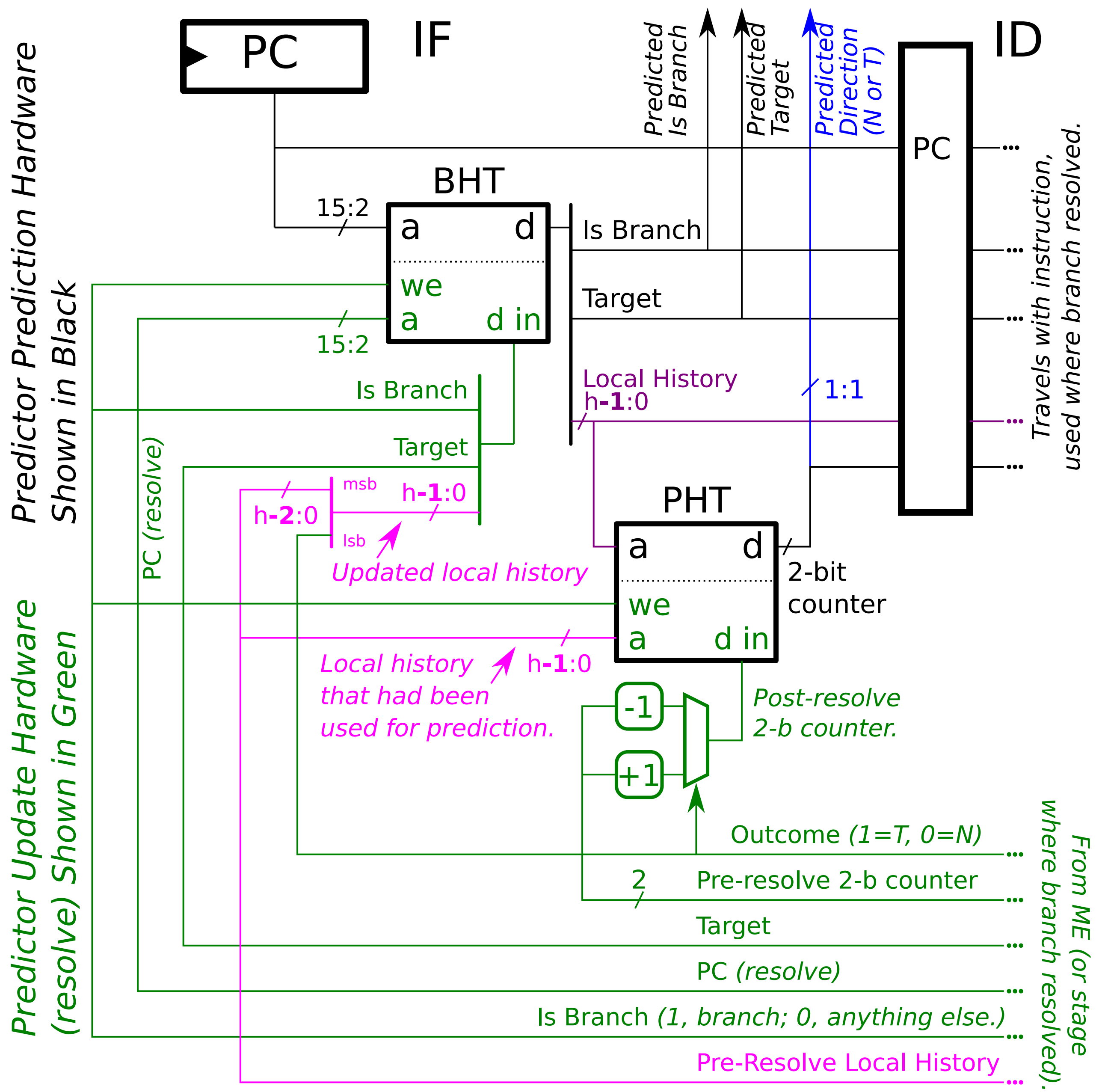
PHT indexed using history concatenated with branch address.

Local, a.k.a., *PAg*.

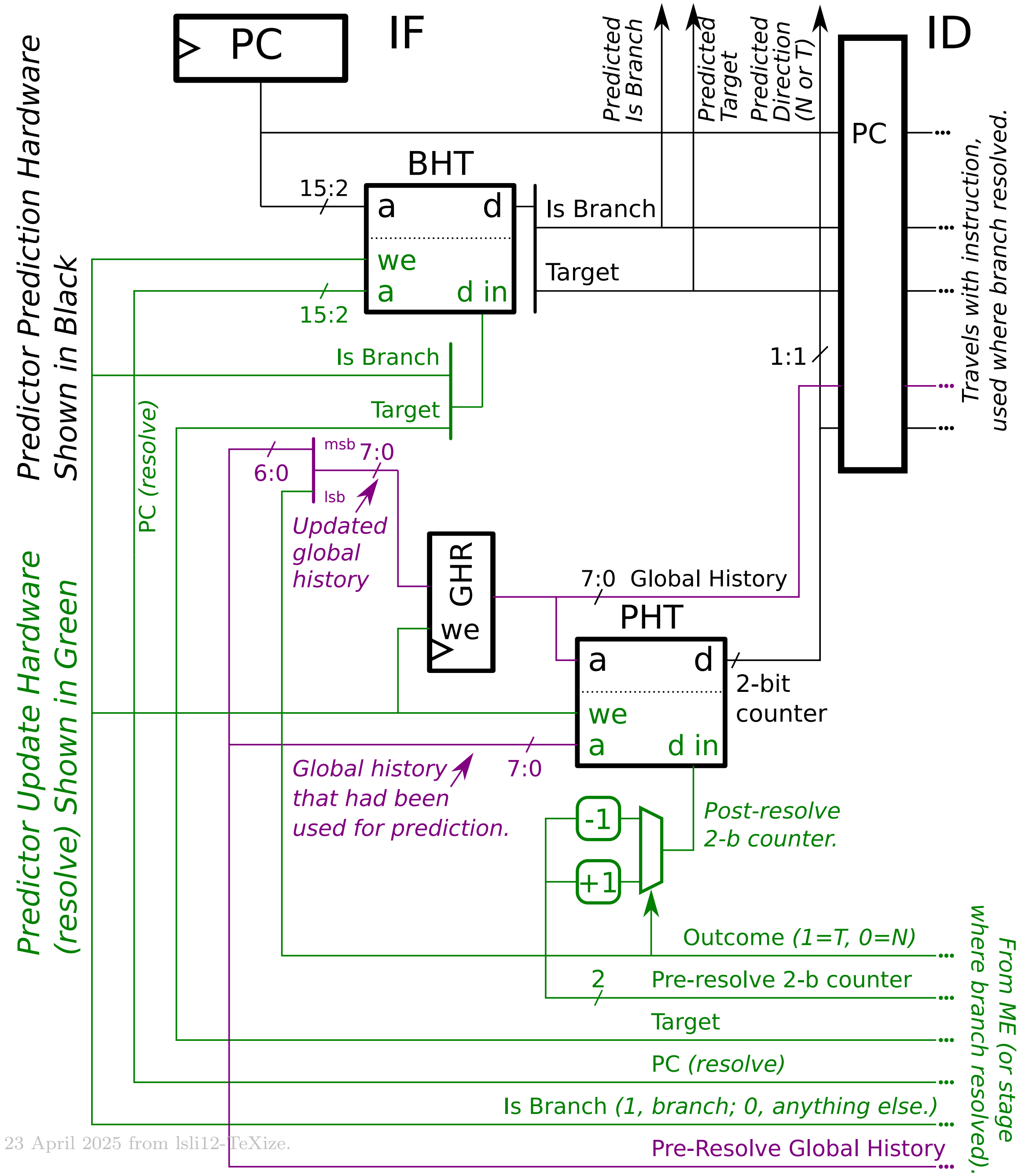
History is local, BHT stores history for each branch.

PHT indexed using history only.

h-Outcome Local History Predictor



Global Predictor



Global History Example

```
# Loop always iterates 4 times.
# Branch below never taken.
 bne  r2, SKIP      N                      N
 add.d f0, f0, f2
SKIP:
 addi  r1, r0, 4
LOOP:
 mul.d f0, f0, f2
 bne   r1, LOOP          T   T   T   N    ... T   T   T   N  ...
 addi  r1, r1, -1
# Cycle                 10  20  30  40  50  110 120 130 140 150
#
# Global History (m=4), X: depends on earlier branches.
# 10  XXXN  Human would predict taken.
# 20  XXNT  Human would predict taken.
# 30  XNTT  Human would predict taken.
# 40  NTTT  Human would predict not taken.
# 50  TTTN
```

Variations on Correlating Predictors

This page is a stub. You can improve it by asking Dr. Koppelman to add material to it.

Three global predictor variations: