Architectural Support for Compiler-Synthesized Dynamic Branch Prediction Strategies: Rationale and Initial Results

David I. August Daniel A. Connors
John C. Gyllenhaal Wen-mei W. Hwu

University of Illinois at Urbana/Champaign
IMPACT Research Group

Conventional Branch Prediction Techniques

	Hardware	Compiler
Static	Uninteresting	Prediction bit, BTFNT
Dynamic	2-bit Counter, 2-Level	Unexplored, Focus of this Work

- Compiler/Static branch prediction
  - Compiler controlled - compile time decision
  - (+) No hardware limits - every branch has a real prediction
  - (+) Low cost
  - (-) Prediction cannot vary during program execution
  - (-) Performs poorly for unbiased branches
- Hardware/Dynamic branch prediction
  - Hardware controlled - state machine makes prediction
  - (+) Accuracy - varies during run-time
  - (-) Hardware has diminishing returns
  - (-) Area, power

Compiler-Synthesized Dynamic Branch Prediction

- Compiler/Dynamic branch prediction
  - Program controlled
    - Compiler inserted code can generate prediction information
    - Take advantage of naturally occuring correlation
  - Sources of information
    - Program state - register values, memory contents
    - Program analysis - symbolic analysis, value prediction
  - Potential benefits
    - (CS) Prediction can vary during program execution
    - (CS) Increased accuracy for unbiased branches
    - (HD) Compiler can perform detailed analysis of program
    - (HD) Effective when there is no correlation among branches

Pipeline Timing of Compiler-Synthesized Dynamic Branch Prediction

- Branch prediction allows instruction fetching to continue speculatively until the branch direction is computed.
- Compiler-Synthesized Dynamic Branch Prediction (CSDBP) computes predictions early in the instruction stream.
- These predictions are forwarded to the branch hardware.

Stage	Instruction
Fetch	Branch p1, DEST
Decode	p1 = Condition
Execute	
Write Back	

Stage	Instruction
Fetch	Branch p1, DEST
Decode	p1 = Condition
Execute	
Write Back	p1 = Prediction

Early Resolved Branches

- *Early-resolved:* the branch condition has been computed early in the instruction stream and is ready when the branch direction is needed.
- Early-resolved branches can be perfectly “predicted”. [RS-6000]
- The CSDBP architecture should take advantage of early-resolved branches.

Stage	Instruction
Fetch	Branch p1, DEST
Decode	p1 = Condition
Execute	
Write Back	

Stage	Instruction
Fetch	Branch p1, DEST
Decode	
Execute	
Write Back	p1 = Condition

- Percent of dynamically encountered branches which are early-resolved:

2 Cycles	62%
4 Cycles	40%
6 Cycles	30%
8 Cycles	24%

Architectural Support for Compiler-Synthesized Branch Prediction

- Predication
  - Conditional execution based upon boolean input
  - Allows the removal of branches and their mispredictions
  - Example:

Before Predication

br_eq r1,r2	
add r1, r1, 1	sub r1, r1, 1
jump DEST	jump DEST

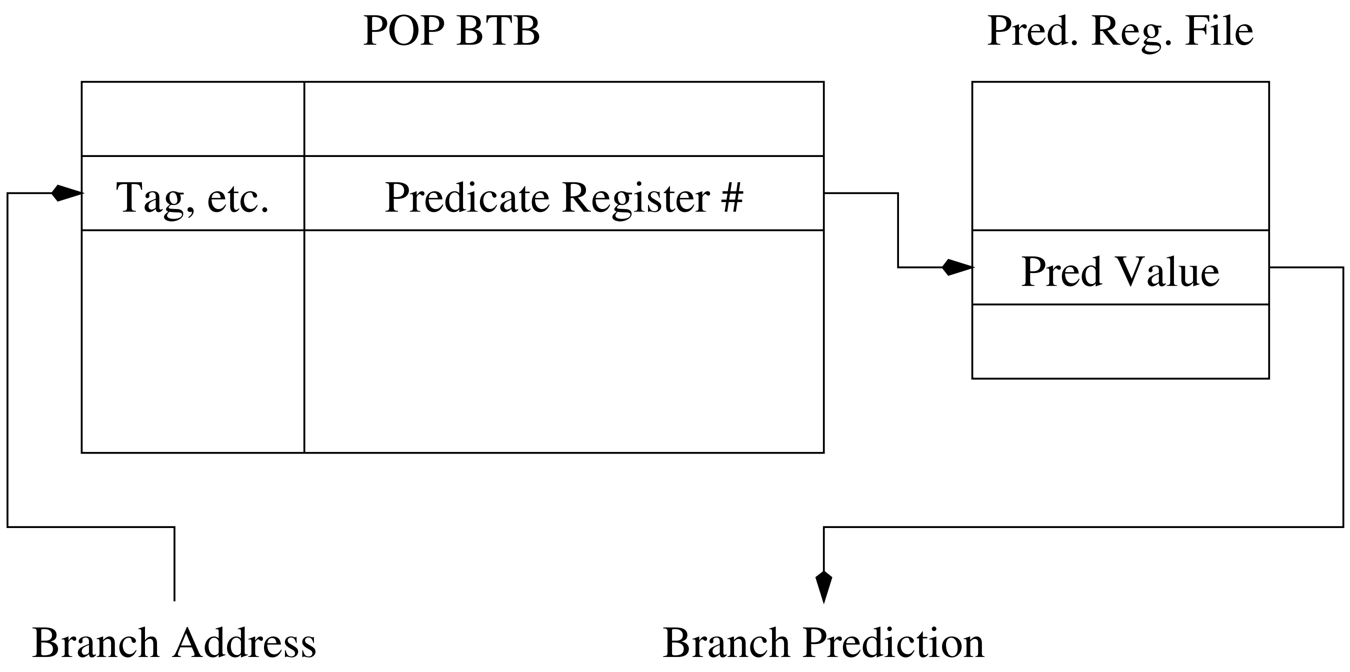
After Predication

pred_eq p1$_{ut}$,p2$_{uf}$,r1,r2	
add r1, r1, 1 (p1)	sub r1, r1, 1 (p2)
jump DEST	

- CSDBP through Predicates
  - Predictions are relayed to the branch hardware through the predicate register file.
  - Instructions which are used to set predicate registers can also be used to compute predictions.

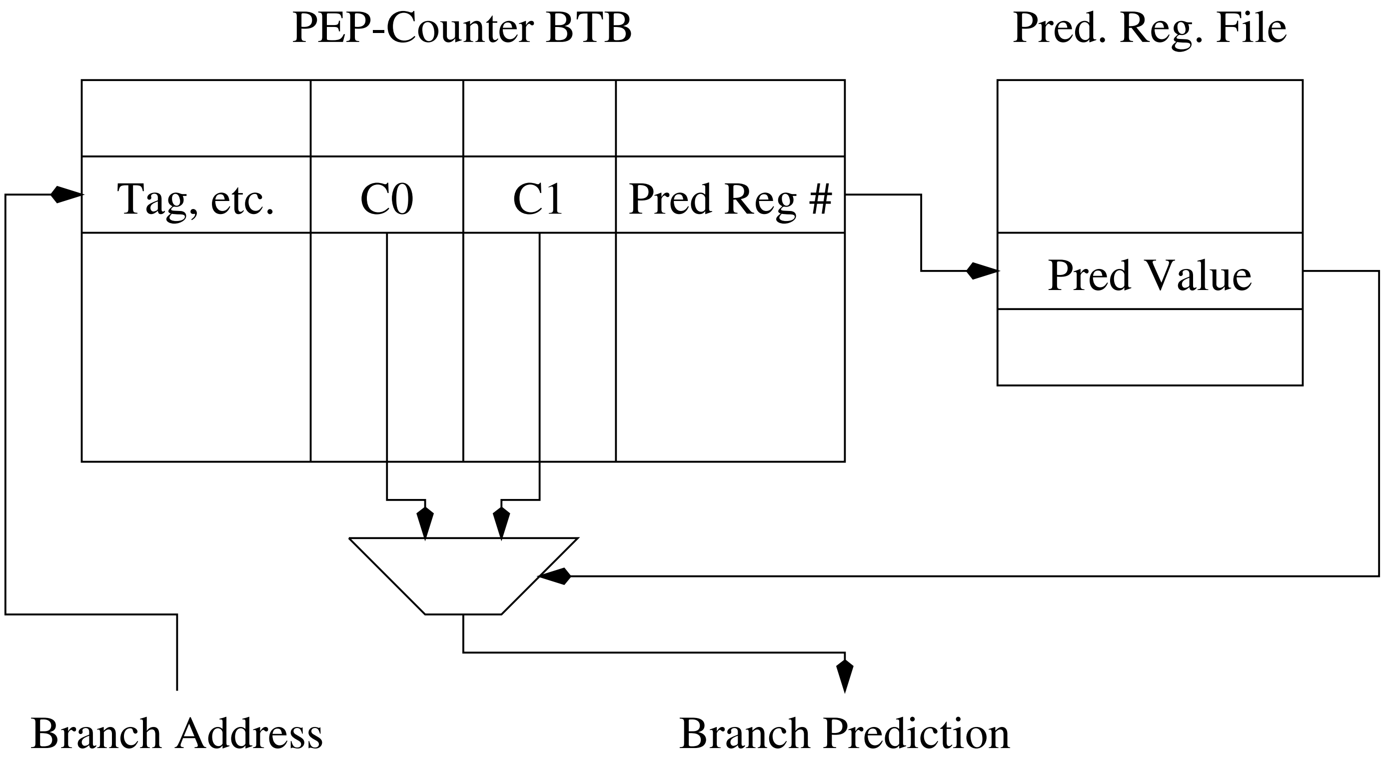
Conceptual Design of the POP Scheme

- Predication Only Prediction (POP) is a pure compiler-synthesized branch predictor.
- The prediction is the value of the branch's condition predicate sampled early.

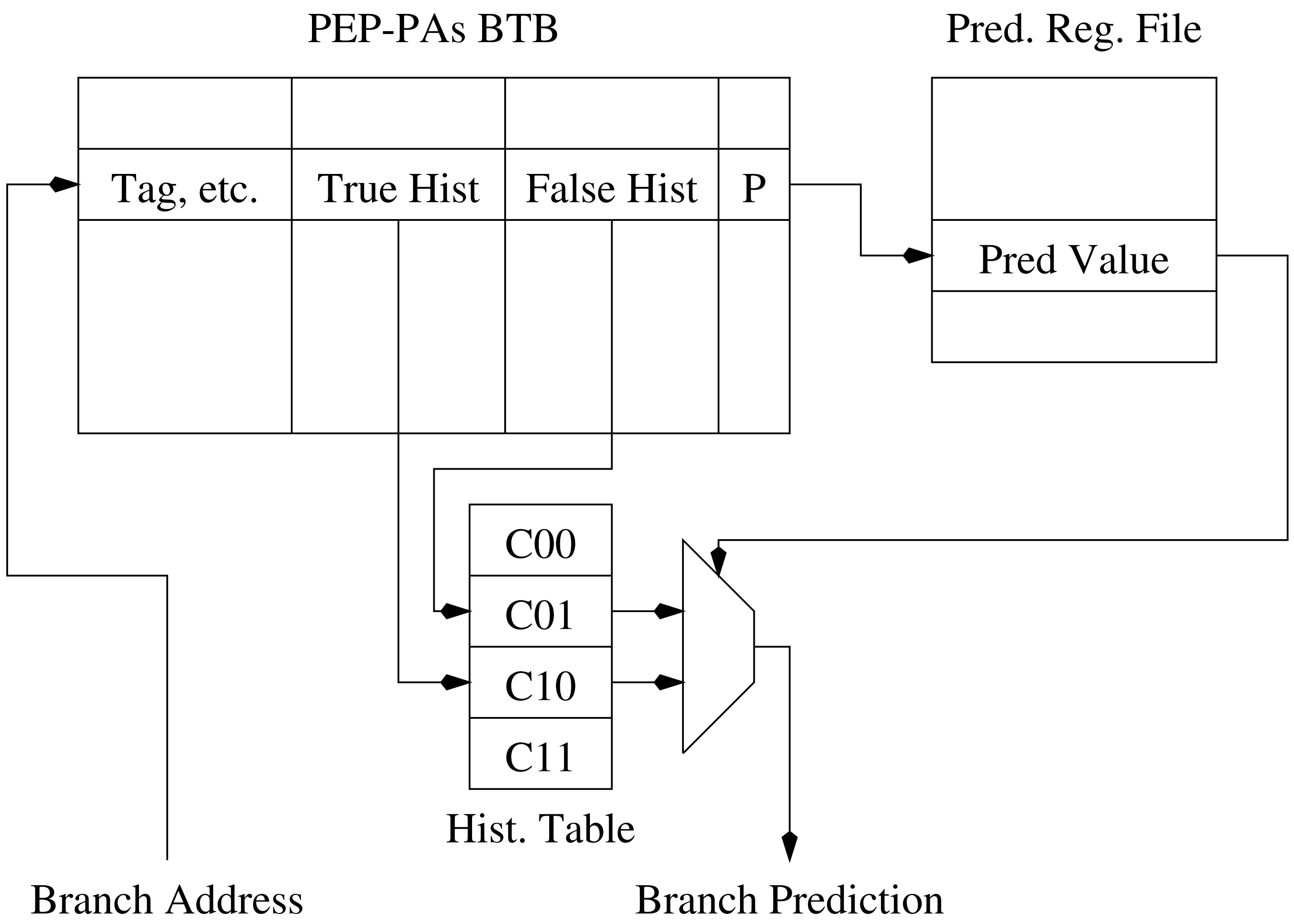


Conceptual Design of the PEP-Counter Scheme

Predication Enhanced Prediction (PEP) uses the branch's condition predicate to enhance conventional branch prediction hardware.



Conceptual Design of the PEP-PAs Scheme

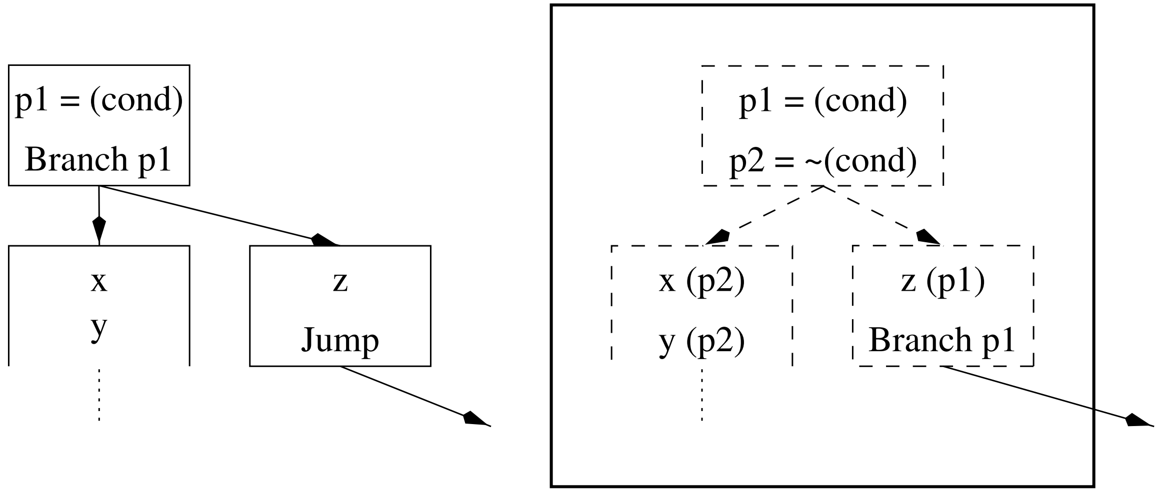


Opportunities for Predicate Based Branch Prediction

- Previous Predicate Value
- Partially Resolved Conditions
- Restoring Desirable Control Flow Characteristics
- Branch Correlation
- Architecture State Correlation
- Prediction by Promotion
- Delaying Branches

Restoring Desirable Control Flow Characteristics

- Some branches must remain after if-conversion due to resource and hazard constraints.
- Remaining branches are predicated.
- The condition on these branches becomes the logical-AND of its original condition and its predicate.
- Use the removed branch's condition as a prediction for remaining branch.



Partially Resolved Conditions

- Many times the condition of a branch is composed of multiple AND/OR operations to the same predicate.
- A perfect prediction can be made for some early values of these predicates.
- Example: In grep's "advance" function, predicate defines on a single branch occur up to 15 cycles early. 49.4% of all mispredictions are removed by this technique.
- May be applicable to OS code which contains many tests.

1	A	pred_eq	$p1_U$,,r1,0
2	B	pred_eq	$p1_{OR}$,,r2,0
3	C	pred_eq	$p1_{OR}$,,r3,0
4	D	pred_eq	$p1_{OR}$,,r4,0
5	E	pred_eq	$p1_{OR}$,,r5,0
6	F	jump	DEST (p1)

Experimental Results

- *Goal*: to measure the amount of existing correlation between predicates and branches.
- The experiments were run on code generated by the IMPACT compiler to set a baseline in performance for future work.
- Architecture:
  - 4-issue in-order superscalar processor
  - 64 integer registers
  - 64 floating-point registers
  - 64 predicate registers
  - HP PA-RISC 7100 instruction latencies
  - 1024 BTB entries, 12 bit history, 16 history tables

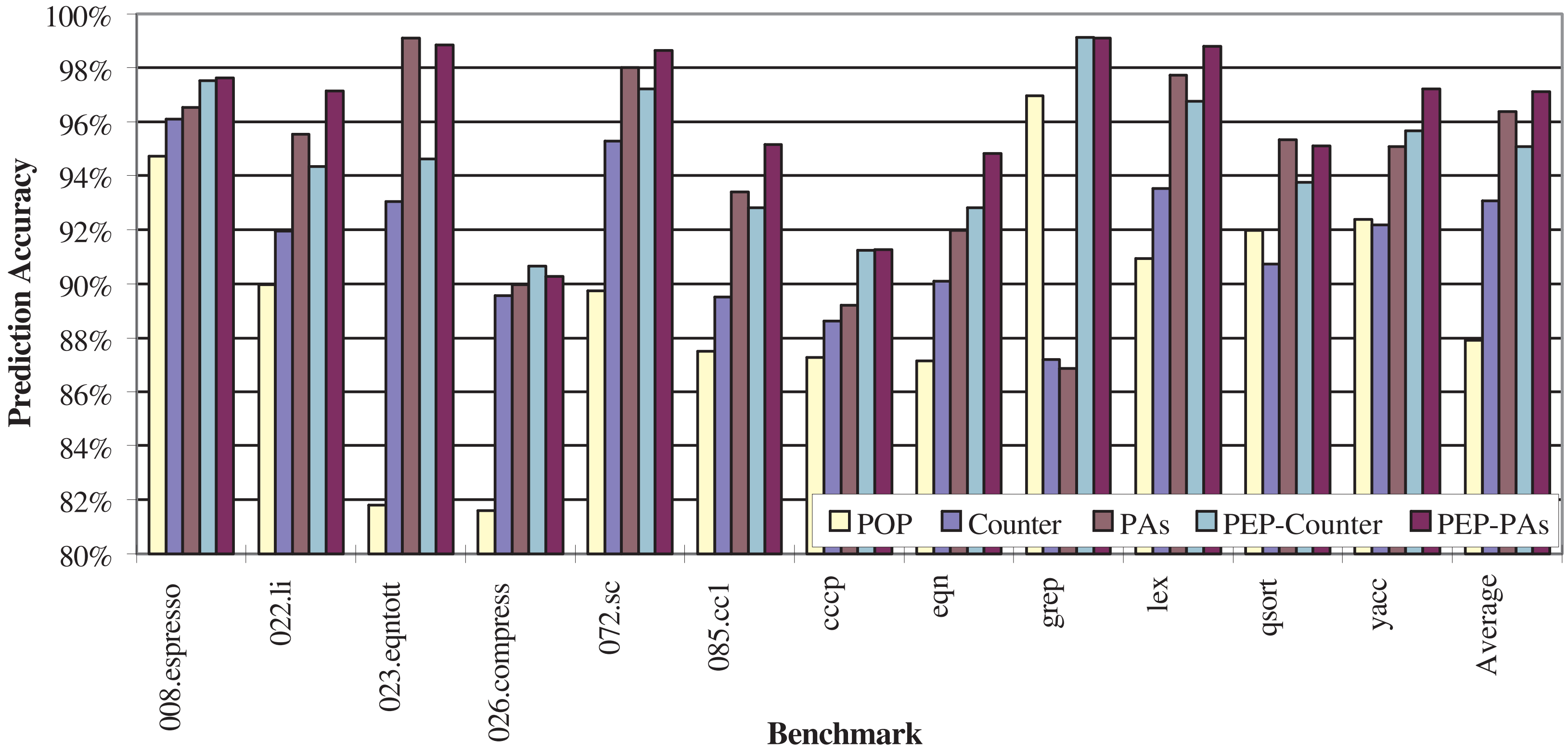
Amount of Correlation

Predicate sampled 4 cycles before branch executes

Benchmark	Early-Resolved	POP Prediction Rate
008.espresso	74%	94.8%
022.li	11%	90.0%
023.eqntott	15%	81.8%
026.compress	39%	81.6%
072.sc	34%	89.7%
085.cc1	42%	87.5%
cccp	34%	87.2%
cmp	51%	87.1%
eqn	48%	87.0%
grep	66%	96.9%
lex	47%	90.9%
qsort	18%	92.0%
yacc	45%	92.4%
Average	40%	87.9%

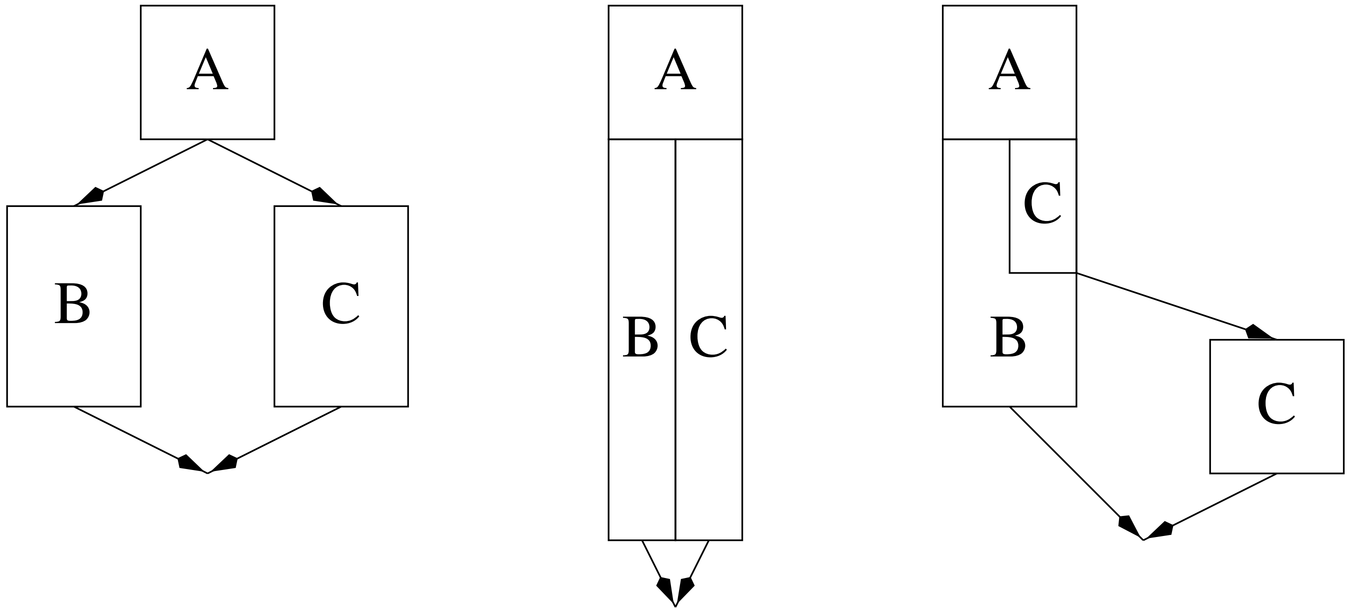
Prediction Rate

Predicate sampled 4 cycles before branch executes



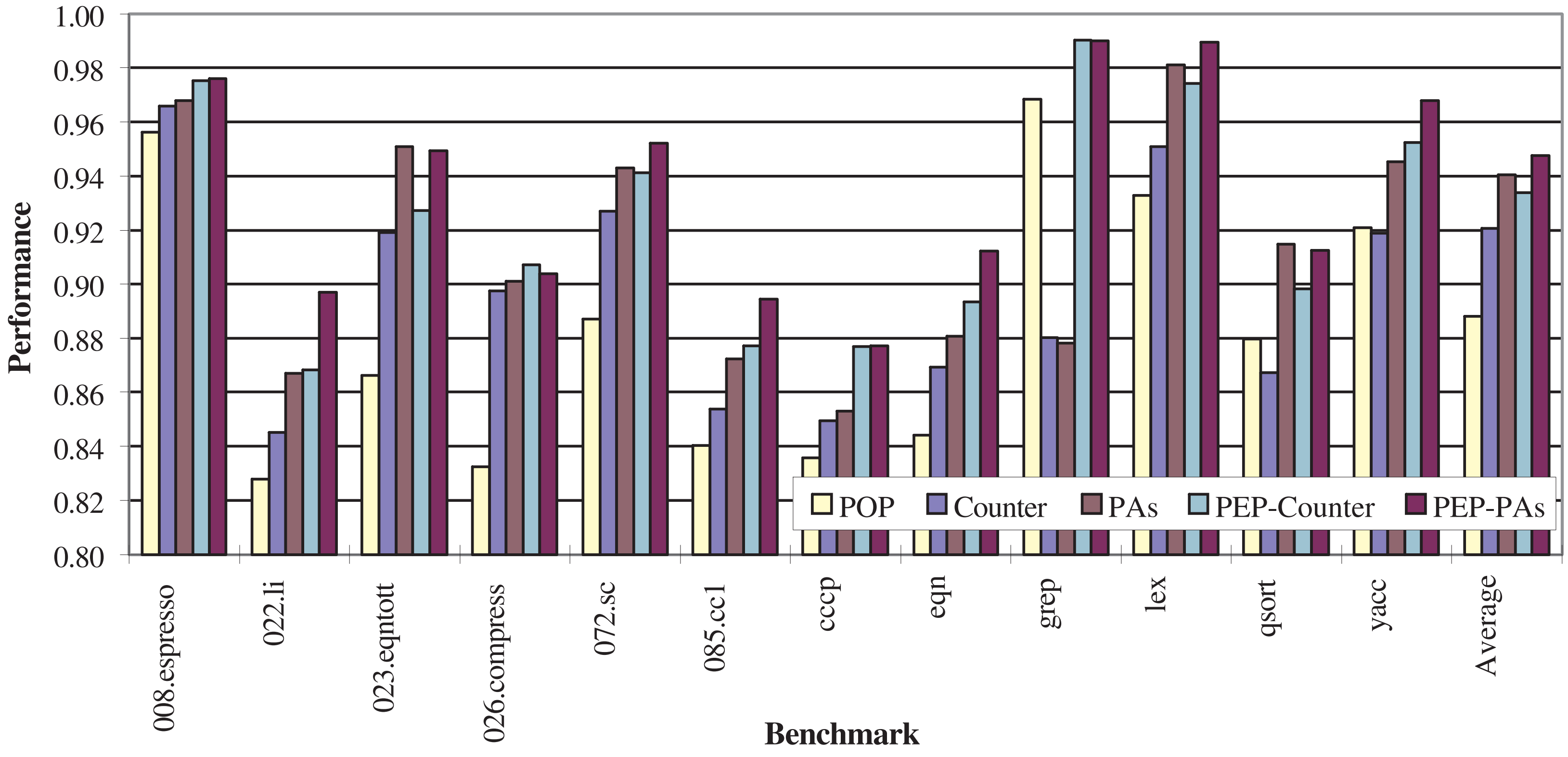
Conclusion and Future Work

- Branches and predicate register state are highly correlated.
- Achieves modest gains with naturally occuring correlation.
- Compiler communicates additional info to the branch predictor.
- Combined if-conversion/CSDBP strategy:

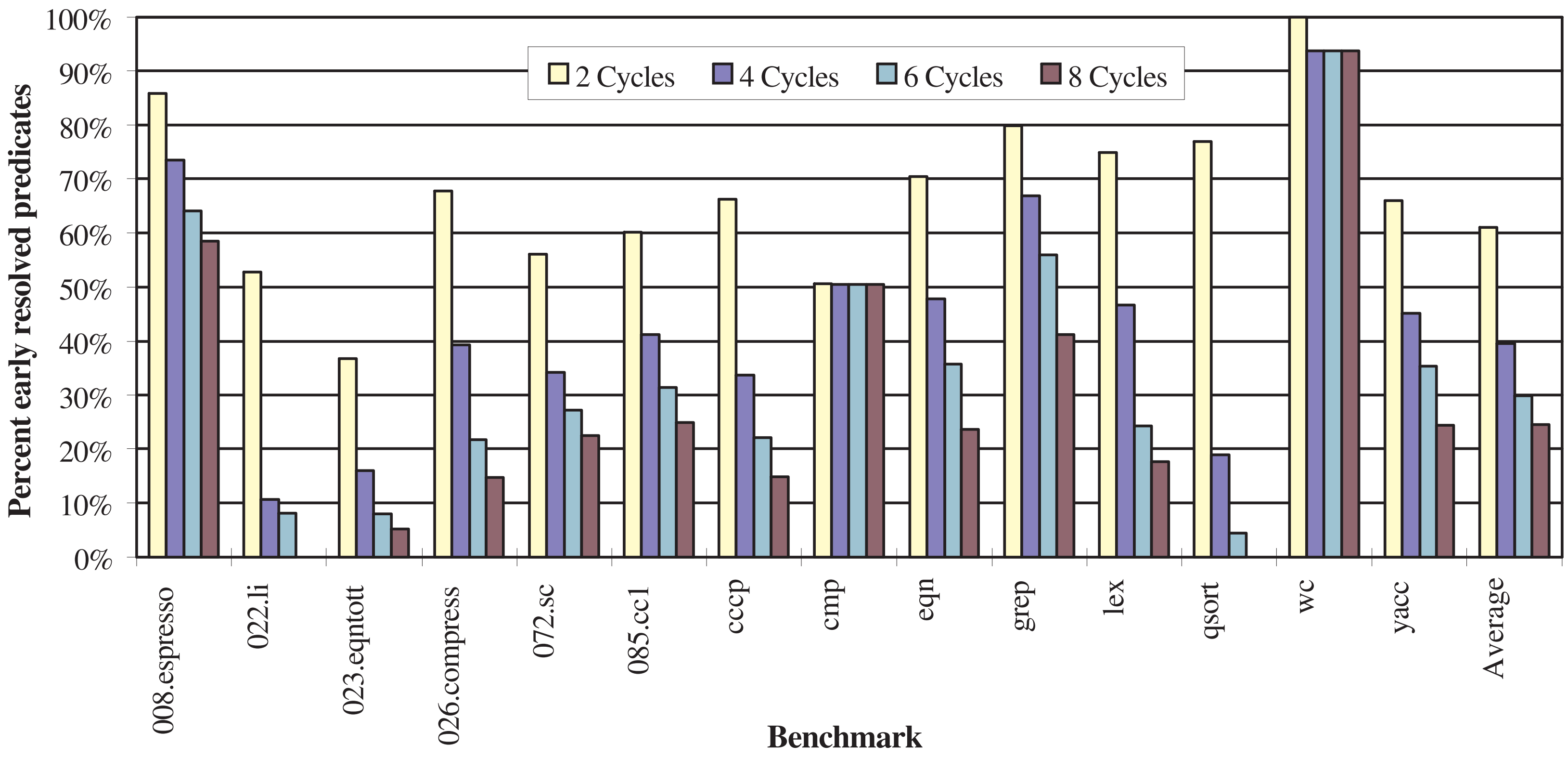


Performance

Predicate sampled 4 cycles before branch executes, 4 cycle miss penalty

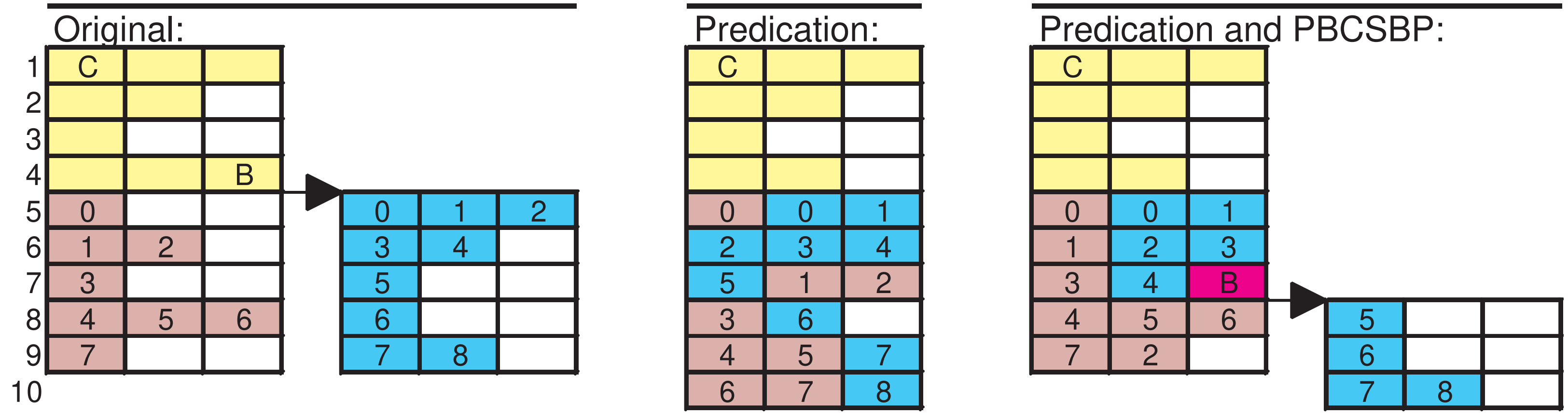


Percent of Branches Early-Resolved



Combined If-Conversion and CSDBP Strategy

Branch Misprediction Penalty = 6 cycles   Taken Frequency = 0.40   Branch Prediction Accuracy = 0.4



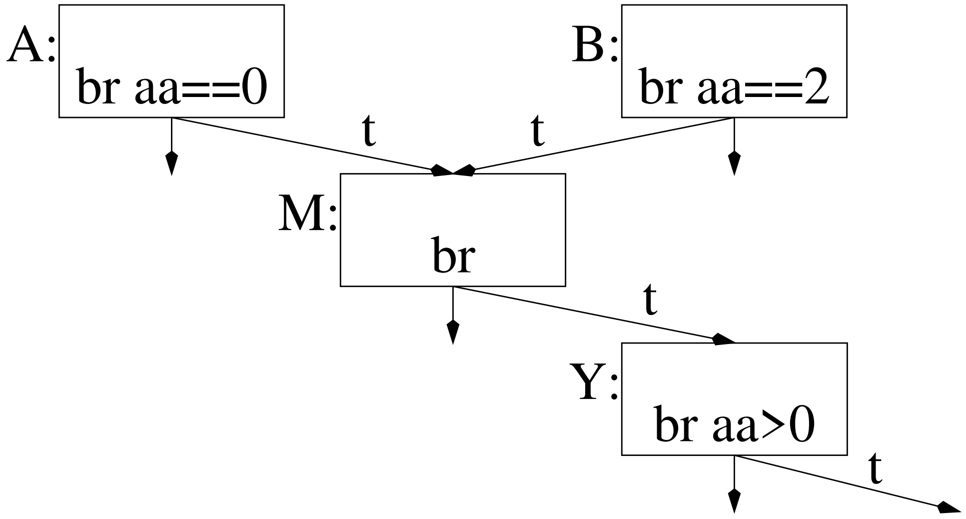
Top	4 x 1.0	4.0
Not Taken	5 x 0.6	3.0
Taken	5 x 0.4	2.0
Penalty	6 x 0.4	2.4
Total		11.4

Path	10.0
Total	10.0

Top	7 x 1.0	7.0
Not Taken	2 x 0.6	1.2
Taken	3 x 0.4	1.2
Total		9.4

Branch Correlation

- Example respectfully taken from Young and Smith ISCA-22.
- Path AMY always yields not taken. Path BMY always yields taken.
- Branch history hardware sees Taken+Taken for both paths and cannot resolve the paths.
- Young and Smith duplicate code to change the history. With CSDBP, paths can be exposed by setting branch predicate to FALSE in A and TRUE in B.



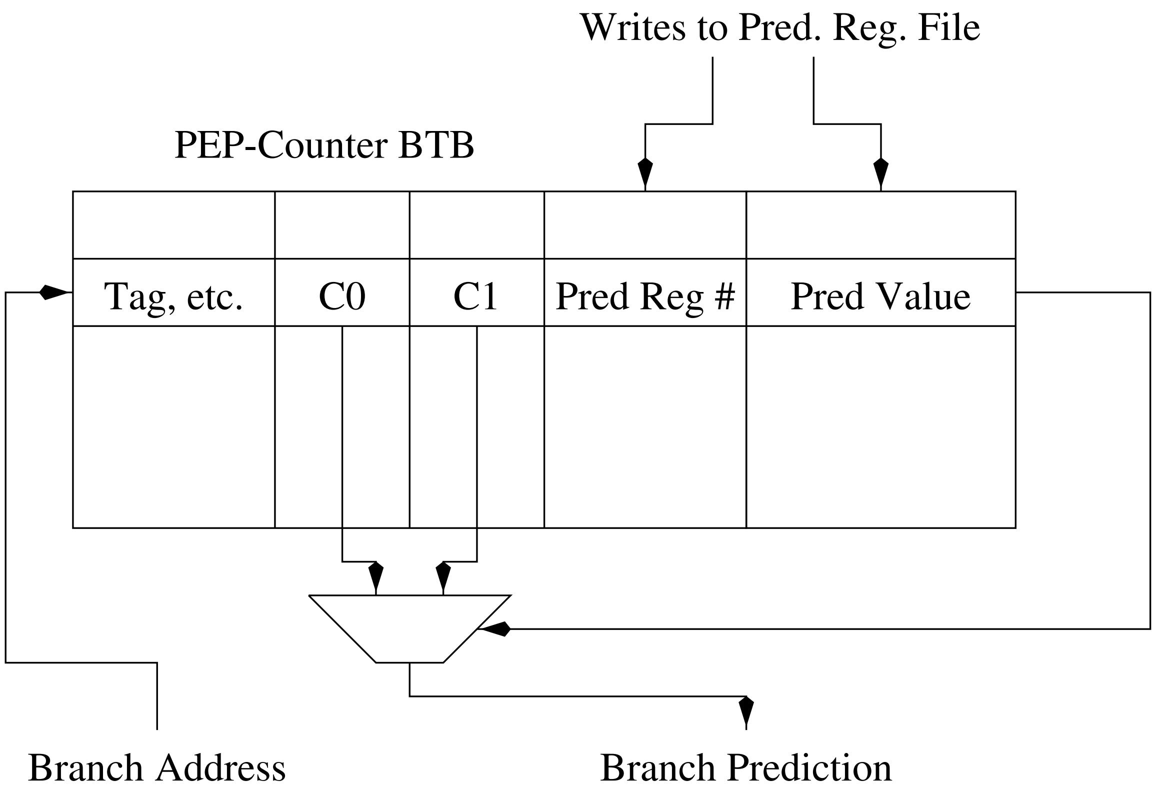
Hardware Cost

Var	Value	Definition
p	64	number of predicate registers
n	1024	number of BTB entries
x	43	bits in standard BTB entry (target address, tag...)
h	12	history register size
t	16	number of history tables

Scheme	Cost Expression	Cost
POP	$n(x + \log_2 p)$	50176
Counter	$n(x + 2)$	46080
PEP-C	$n(x + 4 + \log_2 p)$	54272
PAs	$n(x + h) + 2t(2^h)$	187392
PEP-PAs	$n(x + 2h + \log_2 p) + 2t(2^h)$	205824

Prediction Generation Timing

- Store predicate value in BTB:



- Access BTB entry based upon previous address.