Reducing the Branch Penalty of Mispredicted Short Forward Branches

Gary Tyson, Matthew Farrens,
Kevin Rich and Andrew Pleszkun

Technical Report No. CSE-95-7
Computer Science Department
University of California, Davis, CA 95616
tel: (916) 752-7002, fax: (916) 752-4767
{tyson,farrens,rich}@cs.ucdavis.edu,
arp@tosca.colorado.edu

August 8, 1995

Abstract

Many new high-performance microprocessor designs are incorporating much shorter clocks, deeper pipelines and increased support for multiple instruction issue. The complexities of superscalar issue, however, often require the addition of extra pipeline stages before the execution stage. Having multi-cycle decode and issue stages intensifies the problem of pipeline bubbles due to changes in control flow - while branch prediction can keep the pipeline full of instructions when a branch outcome is correctly predicted, there can be a substantial penalty for flushing instructions and then restoring the proper sequence of instructions on a misprediction.

In this paper, we show that a large percentage of cycles lost due to incorrectly predicted branches are caused by short forward branches, we examine the inefficiencies of pipeline flushing found in current prediction implementations, and we propose some new techniques which can reduce the branch penalty for a significant portion of mispredicted branches. Our simulation results indicate that a 5-15% reduction in the branch penalty can be achieved currently, and savings of 19-30% are possible in the future (as basic block sizes increase).

Reducing the Branch Penalty of Mispredicted Short Forward Branches

Gary Tyson

Department of Computer Science
University of California, Riverside
Riverside, CA
(tyson@cs.ucr.edu)

Matthew Farrens
Kevin Rich

Computer Science Department
University of California, Davis
Davis, CA 95616
(farrens@cs.ucdavis.edu)
(rich@cs.ucdavis.edu)

Andrew R. Pleszkun

Department of Electrical
and Computer Engineering
University of Colorado-Boulder
Boulder, CO 80309-0425
(arp@tosca.colorado.edu)

Abstract

Many new high-performance microprocessor designs are incorporating much shorter clocks, deeper pipelines and increased support for multiple instruction issue. The complexities of superscalar issue, however, often require the addition of extra pipeline stages before the execution stage. Having multi-cycle decode and issue stages intensifies the problem of pipeline bubbles due to changes in control flow - while branch prediction can keep the pipeline full of instructions when a branch outcome is correctly predicted, there can be a substantial penalty for flushing instructions and then restoring the proper sequence of instructions on a misprediction.

In this paper, we show that a large percentage of cycles lost due to incorrectly predicted branches are caused by short forward branches, we examine the inefficiencies of pipeline flushing found in current prediction implementations, and we propose some new techniques which can reduce the branch penalty for a significant portion of mispredicted branches. Our simulation results indicate that a 5-15% reduction in the branch penalty can be achieved currently, and savings of 19-30% are possible in the future (as basic block sizes increase).

1. Introduction

The negative performance impact of changes in control flow on pipelined processors is well-known and well-documented. Put simply, the higher the degree of pipelining in a machine the more difficult it becomes to keep the pipeline full of useful instructions. In the early pipelined RISC processors, the penalty on a branch was small (due to the organization and bypass paths found in a simple pipeline) and the delayed branch concept could be used effectively to minimize the impact of a single cycle branch penalty. However, as both pipeline depths and issue widths continue to increase, much more sophisticated schemes are needed in order to keep the pipeline full.

A myriad of techniques for dealing with this problem have been proposed, ranging from hardware techniques (like conditional execution and highly accurate branch prediction) to software techniques (basic block enlargement, boosting, predicated transformation, etc. [BaGS94]) For example, as superscalar processor designs have begun to increase both the number of instructions that can be issued per machine cycle and the depth of the pipeline, the importance of accurate branch prediction strategies has soared [FiFr92, LeS84, PaS92, Smit81, YeP93]. In a machine that can issue 3 instructions per cycle and has 4 pipeline stages before the branch condition evaluation point, there are as many as 12 instructions that have been fetched and are being processed when the branch condition is evaluated. Currently, if the branch was predicted incorrectly, this entire set of instructions is flushed and the fetch unit starts fetching from the branch target. But what if the branch target is already in the pipeline? How many cycles can be saved by identifying this fact and not flushing the entire set?

In this paper we will present an examination of this question, and some alternate strategies for handling one category of mispredicted branches, those short forward branches that are predicted to be *not taken*. We will show that by carefully handling this single type of misprediction, a non-trivial portion of total cycles lost due to branch misprediction (the **branch penalty**) can be eliminated. In Section 2 we present the pipeline structure of a modern hypothetical processor design to illustrate the potential of this new branch misprediction handling model. Then, Section 3 presents the characteristics of branch activity, where it will be shown that there are many short forward branches. Section 4 analyzes the actual performance of this approach on the SPEC 92 benchmark suite, Section 5 presents a possible implementation of the model, and Section 6 proposes a new hybrid branch prediction scheme to take advantage of the correction model. Finally, Section 7 presents our conclusions and some areas of future work.

2. Pipeline Structure

The pipeline structure of current microprocessor designs differs from the early RISC processors in several respects. The number of pipeline stages has been increasing steadily as more sophisticated architectures are developed, and because of the complexities of superscalar issue, these new stages tend to be earlier in the pipeline (before the execution stage). The existence of multi-cycle decode and issue stages has a definite negative impact on branch handling by increasing the branch penalty when the prediction is not correct.

A closer examination of what is occurring in the pipeline reveals room for improvement in how mispredicted branches are handled. Figure 1 shows a hypothetical machine with a 7 stage pipeline that can issue 4 instructions per cycle, and a small sample program being executed on the machine. In this example, the branch at location **10c** has been predicted *not taken*, and the branch at **12c** has been predicted *taken*. If the prediction of the branch at **10c** proves to be incorrect, current implementations simply invalidate all the instructions in the fetch, decode and issue stages (stages 1-4) and begin fetching from the correct target address.

However, in the case of instructions that branch forward a short distance, many of the instructions are being invalidated needlessly, only to be refetched immediately after being invalidated. This example shows that of the 12 instructions past the branch that have been fetched, 10 of them are instructions that will be executed. Flushing the entire Fetch/Decode/Issue pipeline is clearly not optimal, since stages 1 and 2 are being flushed unnecessarily.

This example demonstrates another interesting point about the savings involved in not flushing the entire pipeline. Let us assume that not only **10c**, but also **12c** is incorrectly predicted. In this case, we are allowing some instructions to remain in the pipeline that must later be flushed (when the second prediction is determined to be false). While this seems to be a dubious savings, there *is* in fact a reduction in branch penalty - if the entire pipeline is flushed when **10c** was determined to be incorrect, the same instructions (**118-12c**) would have to be reloaded and the same (incorrect) prediction would be made on **12c**, resulting in 2 flushes with the conventional restoration logic. The bottom line is that it does not matter whether the instructions remaining in the pipeline are correct, only whether or not they will be immediately refetched.

This example has demonstrated the *potential* for reducing the cycles lost to incorrectly predicted short forward branches. However, it only works for mispredicted short forward branches. The next logical question is, how much of the total branch misprediction penalty is due to this category of branch?

Source Code	Addr	Target Instructions			
int x=0, y[100], z[100], i, max, sum;	**100**		move	R2, R0	; R2 = (i = 0);
main() {	**104**		move	R3, R0	; R3 = (x = 0);
	108		movi	R6,#100	; R6 = (100);
for (i=0; i<100 ; i++) {		L32:			
if (x>max) {	**10c**		brle	R3, R4, L33	; if (x>max)
max = x;	**110**		move	R4, R3	; R4 = (max = x);
sum += x;	**114**		add	R5, R3, R5	; R5 = (sum += x);
}		L33:			
	118		mul	R9, R2, 4	;
x = y[i] + z[i];	**11c**		lw	R10, z(R9)	; R10 = z[i]
}	**120**		lw	R11, y(R9)	; R11 = y[i]
}	**124**		addu	R12, R10, R11	; x = R10 + R11
	128		addu	R2, R2, 1	; i++;
	12c		blt	R2, R6, L32	; loop while i<100

Fetch	Decode	Issue		Execute		Commit
10c	120	*110*	**100**	**...**	**...**	**...**
110	124	*114*	**104**	**...**	**...**	**...**
114	128	118	**108**	**...**	**...**	**...**
118	12c	11c	**10c**	**...**	**...**	**...**
1	**2**	**3**	**4**	**5**	**6**	**7**

Figure 1: Pipeline Structure of Example Processor and Sample Code Excerpt

3. Branch Behavior Statistics

The benchmarks used for this study were the fourteen floating point and five of the integer programs from the SPEC 92 suite of programs. Each program was compiled on an Alpha-based DEC 3000/400 workstation, using the native compiler and **-O2 -non_shared** compiler flags. The ATOM [SrWa94] toolkit was used to generate and help analyze the data. Table 1 shows the benchmark programs and the inputs used, and the total number of instructions, branches, and jumps executed, respectively.

The number of branches in Table 1 are further broken down by branch distance in Figure 2, which shows the percentage of branches taken for each distance (in instructions) between the branch instruction and the target address. Note that for both integer and floating point applications, short forward branches constitute a high percentage of all branches (cumulatively approaching 25%).

Table 1: SPEC Benchmark Information

Benchmark	Input	Instructions	Branches	Jumps
alvinn	*	3,554,909,199	167,524,380	30,274,280
doduc	doducin.in	1,149,864,381	84,622,229	13,463,476
ear	args.ref	17,005,800,990	896,667,195	481,185,420
fpppp	natoms	4,333,190,502	115,692,876	6,508,987
hydro2d	hydro2d.in	5,682,547,494	347,131,563	9,876,849
mdljdp2	input.file	6,856,424,748	320,407,455	71,051
mdljsp2	input.file	2,898,940,578	352,481,636	1,024,674
nasa7	*	6,128,388,226	165,573,273	23,201,959
ora	params	6,036,097,727	365,260,354	87,096,763
spice2g6	greycode.in	16,148,172,367	1,933,779,718	93,549,064
su2cor	su2cor.in	4,776,761,988	178,137,872	30,247,847
swm256	swm256.in	11,037,397,686	182,031,528	407,016
tomcatv	*	899,655,110	30,183,243	27,571
wave5	*	3,554,909,199	167,524,380	30,274,280
compress	in	92,628,682	12,379,188	502,775
espresso	bca.in	424,397,814	74,371,674	3,603,641
espresso	cps.in	513,006,475	83,519,772	4,051,876
espresso	ti.in	568,262,371	89,791,260	5,083,215
espresso	tial.in	983,531,458	167,375,187	12,731,193
gcc	cexp.i	23,535,507	3,098,517	506,777
gcc	jump.i	143,737,833	6,901,206	1,063,385
gcc	stmt.i	51,359,229	19,426,084	2,947,663
xlisp	li-input.lsp	6,856,424,748	867,585,743	316,747,153
eqntott	int_pri_3.eqn	1,810,542,679	199,198,053	9,669,793
sc	loada1	1,450,169,424	277,556,127	29,920,923
sc	loada2	1,634,276,007	328,405,132	46,912,615
sc	loada3	412,097,081	91,802,679	6,375,876

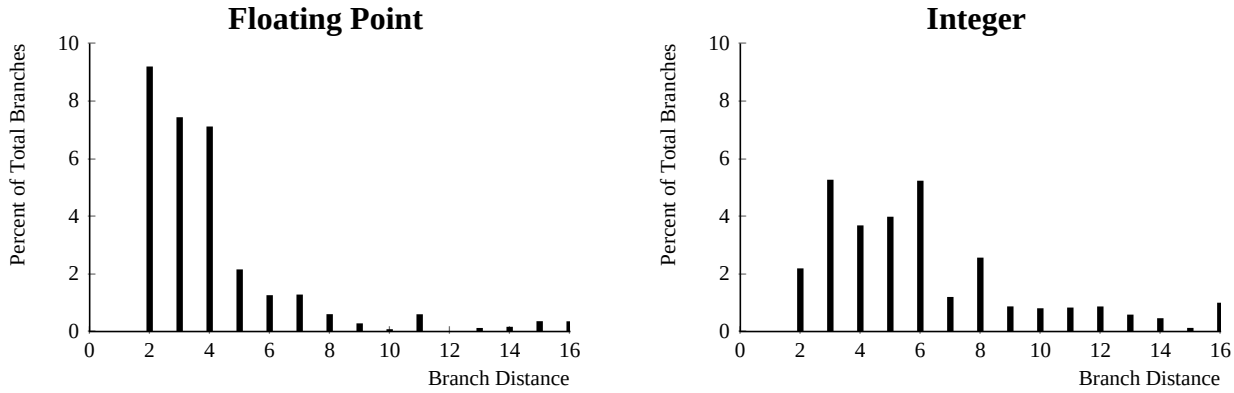

Figure 2: Branch Distribution

Furthermore, it is these short forward branches that have the poorest prediction rate. This can be seen in Figure 3, which shows the average miss-prediction rate for branches of distance less than 2, 2-12, and greater than 12. The four prediction schemes presented in Figure 3 are:

2-bit Simple	A simple predictor using a 2048-entry table of 2-bit counters [Smit81]
3-bit Simple	A simple predictor using a 2048-entry table of 3-bit counters [Smit81]
2-bit Adaptive	An adaptive 2-level predictor using a 13-bit pattern history table and a 2048-entry table of 2-bit counters [YeP91]
3-bit Adaptive	An adaptive 2-level predictor using a 13-bit pattern history table and a 2048-entry table of 3-bit counters [YeP91]

Looking at the relationship between branch penalty and branch distance reveals that a very large portion of the penalty comes from these branches. This can be seen by looking at Figure 2 (which shows that there are a large number of branches that are of the short forward variety), Figure 3 (which shows that they have the worst prediction rate), and the following equation:

*branch penalty = cost of executing branch * total number of branches executed*
*+ misprediction penalty * number of mispredicted branches*

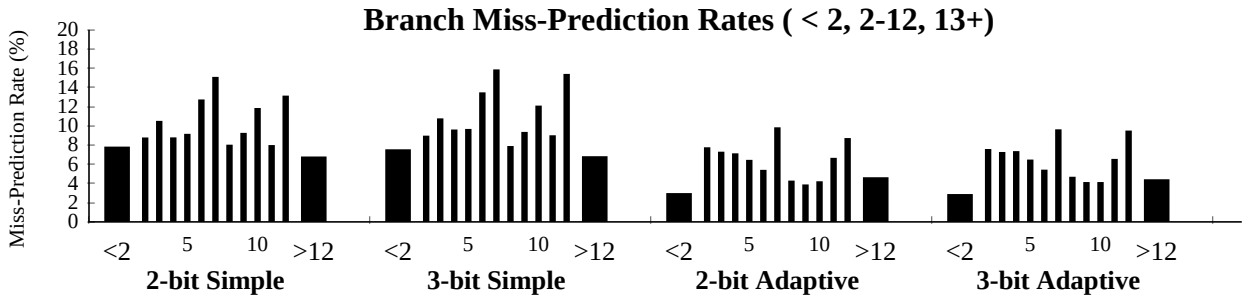

Figure 3: Short Forward Branch Miss-Prediction Rates

The fact that the short forward branches are responsible for so much of the branch penalty implies that there is substantial useful information in the pipeline that is being flushed on a misprediction, and will have to be re-fetched.

4. Simulation Results

In order to measure the actual number of cycles that could be saved by not flushing the pipeline on mispredicted short forward branches, we performed two separate sets of simulations. In the first set we measured the **Potential** reduction in branch penalty cycles. In these simulations the types of all instructions lying between the current branch instruction and the branch target address were ignored, and the branch target was always assumed to be in the pipeline if the branch distance was less than the product of the pre-issue pipeline depth and the issue width. These simulations give a measure of the maximum potential savings, and provide something of an upper bound on the available branch penalty reduction.

While this situation will often occur (as sufficiently large basic blocks become the norm), measurements of existing programs must account for the fact that there are many nested branches in a program, and the chances of the correct target address being somewhere in the pipeline are clearly reduced. Therefore, we ran a second set of simulations in which all control flow instructions (both branches and jumps) residing between the current branch and the branch target were evaluated, predicted, and the instruction fetch directions modified accordingly. Only when the fetch logic would actually have fetched the branch target address was the branch penalty reduced. So, in these simulations, a branch around a jump would result in the branch target address *not* being found in the pipeline, since the jump would be predicted to be taken and the fetch logic would be fetching down a different path when the branch condition was finally evaluated. We called this more realistic set of simulations the **measured** savings.

The results of the simulations are presented in tabular form in Tables 3-6. Tables 3 and 5 show the results of the simulations using the simple 2-bit and 3-bit counters as the predictors, while Tables 4 and 6 contain the results of the simulations using the adaptive 2-bit and 3-bit predictors. In each table, C denotes the number of pipeline stages before and including the issue stage, and W indicates the issue width in instructions. The percentages presented in columns 3-6 are all averages over the entire set of benchmarks, and are calculated in cycles, which is a function of the number of pipeline stages and the issue width. The third column, the *Short Penalty* column, indicates the percentage of the total branch penalty paid by all forward branch instructions of distance less than (C * W). Column 4 shows the percentage reduction in branch penalty for the Potential case, and column 5 shows the measured savings for each configuration. Finally, Column 6 (Best Case) gives the largest potential percentage reduction in branch penalty for all of the benchmark programs for that combination of C and W. This was included to show that

even when the average reduction may be relatively small, some program within the suite received substantial benefits from the approach.

Looking at Table 3, for example, we see that for C=2 and W=8, 44.23% of the total branch penalty is due to mispredicted forward branches with a branch distance less than 16, the potential reduction in this penalty is 8.57%, the measured savings is 3.74%, and one of the benchmarks experienced a 19.49% reduction (the identity of the program is not included in the table).

As expected, the tables reveal that mispredicted short forward branches account for a very large percentage of the total branch penalty. They also show that the potential savings are substantial, but are significantly less than the short penalty. This is because the short penalty includes the effects of incorrectly predicting both taken and not taken branches. Not flushing the pipeline will only reduce the penalty due to branches that were incorrectly predicted to be not taken - it will do nothing to mitigate the effect of incorrectly predicted taken branches. It is also worth emphasizing that currently the branch penalty is a function of C and W. Using this proposed technique, the branch penalty will become instead a function of the branch distance (the distance between the branch and the target). Clearly, as the product of C and W increases and the branch distance stays fixed, the savings will approach the theoretical limit of half of the short penalty[1]. This can be seen in the tables as well - since the bulk of the branch penalty reduction comes from hiding the impact of the very short forward branches (branch distances less than approximately 12), as C*W grows the potential savings can be seen approaching 50% of the total.

The tables also show that the measured savings are much less than the potential savings. One can expect a certain amount of movement towards the potential case in the future, as compiler techniques continue to improve. It is also clear from the "best case" column of the tables that some programs experience very significant (20%-30%) measured reductions in branch penalties.

Finally, it is worth noting that the reduction in branch penalty available depends on the accuracy of the branch predictor. This effect can be seen by looking at Tables 3 and 4, for example, which shows that the potential savings for the simple 2-bit scheme consistently benefits less than the 2-bit adaptive scheme. This is due to the fact that while the adaptive scheme is much better at predicting backward and long forward branches, both schemes are less adept at correctly predicting short forward branches. This increases the relative weight of the short forward penalty and therefore the benefit of reducing it.

---

[1]For 2-bit counter schemes, the branch penalty is split evenly between incorrectly predicted Not Taken and Taken.

Table 3. Simple Branch Predictor using 2-bit Counters

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
2	1	7.66	0.00	0.00	0.01
2	2	23.63	1.37	1.37	11.10
2	3	32.45	4.27	2.25	15.41
2	4	36.32	5.22	2.90	15.41
2	5	39.70	5.65	3.01	15.41
2	6	42.76	7.65	3.24	15.41
2	7	43.53	8.17	3.54	19.49
2	8	44.23	8.57	3.74	19.49
2	9	45.48	9.12	4.15	19.49
3	1	19.12	0.92	0.91	7.40
3	2	32.45	4.40	2.84	14.81
3	3	37.99	7.94	3.66	20.55
3	4	42.76	9.19	4.42	23.27
3	5	43.70	10.07	4.85	23.27
3	6	45.48	12.02	5.48	23.27
3	7	46.94	12.48	5.74	25.99
3	8	48.44	12.86	5.92	25.99
3	9	49.54	13.49	6.24	25.99
4	1	23.63	2.82	1.81	11.11
4	2	36.32	7.12	3.75	16.76
4	3	42.76	10.52	4.82	25.16
4	4	44.23	12.09	5.81	27.20
4	5	46.39	12.84	6.18	27.20
4	6	48.44	14.58	6.72	27.20
4	7	49.74	15.15	7.00	29.24
4	8	50.26	15.69	7.16	29.24
4	9	50.55	16.31	7.50	29.24
5	1	26.28	4.34	2.60	13.33
5	2	39.70	9.12	4.50	20.50
5	3	43.70	12.57	5.85	27.93
5	4	46.39	13.96	6.70	29.56
5	5	48.80	14.83	7.06	29.56
5	6	50.16	16.50	7.55	29.56
5	7	50.50	17.09	7.85	31.19
5	8	52.61	17.58	8.00	31.19
5	9	53.66	18.11	8.28	31.19

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
6	1	32.45	5.50	3.17	15.72
6	2	42.76	10.75	5.17	23.58
6	3	45.48	14.00	6.57	29.77
6	4	48.44	15.43	7.35	31.13
6	5	50.16	16.42	7.70	31.13
6	6	50.55	17.93	8.18	31.13
6	7	53.14	18.46	8.45	32.49
6	8	53.91	19.01	8.61	32.49
6	9	56.20	19.51	8.88	32.49
7	1	34.87	6.90	3.64	17.88
7	2	43.53	12.19	5.86	25.78
7	3	46.94	15.18	7.12	31.09
7	4	49.74	16.68	7.85	32.25
7	5	50.50	17.66	8.22	32.25
7	6	53.14	18.99	8.65	32.25
7	7	53.99	19.59	8.93	33.42
7	8	56.32	20.09	9.08	33.42
7	9	57.30	20.78	9.53	33.42
8	1	36.32	8.08	4.08	20.52
8	2	44.23	13.30	6.39	27.43
8	3	48.44	16.18	7.58	32.07
8	4	50.26	17.70	8.28	33.09
8	5	52.61	18.61	8.62	33.09
8	6	53.91	19.91	9.05	33.10
8	7	56.32	20.47	9.31	34.12
8	8	57.67	21.14	9.63	34.12
8	9	58.03	21.80	10.05	34.12
9	1	37.99	9.09	4.45	22.57
9	2	45.48	14.20	6.82	28.72
9	3	49.54	17.07	7.95	32.84
9	4	50.55	18.53	8.62	33.75
9	5	53.66	19.45	8.96	33.75
9	6	56.20	20.66	9.36	33.75
9	7	57.30	21.36	9.77	34.66
9	8	58.03	22.03	10.09	34.66
9	9	59.07	22.65	10.49	34.66

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

Table 4. Adaptive Branch Predictor using 2-bit Counters

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
2	1	10.48	0.00	0.00	0.01
2	2	28.77	1.79	1.79	19.14
2	3	36.29	4.90	2.81	19.17
2	4	40.26	6.14	3.70	19.17
2	5	42.32	6.59	3.85	19.17
2	6	44.94	7.55	4.09	19.17
2	7	45.62	8.20	4.45	19.80
2	8	46.54	8.56	4.62	19.80
2	9	47.97	8.79	4.77	19.87
3	1	22.99	1.19	1.19	12.76
3	2	36.29	5.29	3.66	25.54
3	3	41.11	8.30	4.60	25.56
3	4	44.94	9.80	5.55	25.56
3	5	45.86	10.43	5.87	26.03
3	6	47.97	11.48	6.37	26.03
3	7	49.47	12.02	6.68	26.39
3	8	50.21	12.40	6.89	26.39
3	9	51.49	12.78	7.09	26.50
4	1	28.77	3.35	2.30	19.16
4	2	40.26	7.74	4.79	28.74
4	3	44.94	10.62	5.84	29.11
4	4	46.54	12.18	6.89	29.11
4	5	48.86	12.76	7.20	29.46
4	6	50.21	13.80	7.70	29.46
4	7	51.67	14.40	8.00	29.70
4	8	52.37	14.78	8.20	29.70
4	9	52.97	15.23	8.43	29.81
5	1	31.89	5.13	3.32	22.99
5	2	42.32	9.62	5.68	30.66
5	3	45.86	12.36	6.86	31.23
5	4	48.86	13.76	7.80	31.23
5	5	50.76	14.46	8.13	31.52
5	6	52.24	15.43	8.59	31.52
5	7	52.87	16.06	8.90	31.68
5	8	54.53	16.42	9.09	31.68
5	9	55.55	16.85	9.29	31.80

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
6	1	36.29	6.48	4.05	25.55
6	2	44.94	11.03	6.38	32.18
6	3	47.97	13.60	7.58	32.65
6	4	50.21	15.01	8.47	32.66
6	5	52.24	15.75	8.80	32.89
6	6	52.97	16.68	9.26	32.89
6	7	55.33	17.25	9.53	33.00
6	8	55.88	17.64	9.75	33.00
6	9	58.85	18.06	9.97	33.13
7	1	38.84	7.71	4.64	27.38
7	2	45.62	12.22	7.03	33.26
7	3	49.47	14.62	8.17	33.66
7	4	51.67	16.00	8.99	33.67
7	5	52.87	16.78	9.35	33.87
7	6	55.33	17.63	9.75	33.88
7	7	55.99	18.23	10.08	33.94
7	8	58.97	18.59	10.28	33.94
7	9	60.02	19.24	10.70	34.09
8	1	40.26	8.80	5.17	28.75
8	2	46.54	13.15	7.54	34.07
8	3	50.21	15.49	8.65	34.43
8	4	52.37	16.83	9.43	34.43
8	5	54.53	17.58	9.77	34.61
8	6	55.88	18.43	10.21	34.61
8	7	58.97	18.99	10.50	34.65
8	8	60.68	19.56	10.88	34.65
8	9	61.03	20.19	11.28	34.81
9	1	41.11	9.72	5.63	29.82
9	2	47.97	13.92	7.97	34.70
9	3	51.49	16.21	9.03	35.02
9	4	52.97	17.52	9.79	35.02
9	5	55.55	18.26	10.13	35.18
9	6	58.85	19.09	10.56	35.19
9	7	60.02	19.81	11.01	35.20
9	8	61.03	20.41	11.38	35.21
9	9	62.08	20.99	11.74	35.37

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

Table 5. Simple Branch Predictor using 3-bit Counters

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
2	1	7.86	0.00	0.00	0.01
2	2	23.96	1.20	1.19	11.34
2	3	32.92	3.85	2.12	15.25
2	4	36.70	4.65	2.66	15.25
2	5	39.89	5.07	2.75	15.25
2	6	43.33	7.06	2.99	15.25
2	7	44.14	7.50	3.21	17.20
2	8	44.91	7.92	3.40	17.20
2	9	46.16	8.44	3.79	17.21
3	1	19.52	0.80	0.80	7.56
3	2	32.92	3.90	2.57	15.12
3	3	38.16	7.28	3.41	20.33
3	4	43.33	8.37	4.04	21.64
3	5	44.32	9.24	4.47	21.64
3	6	46.16	11.22	5.10	21.64
3	7	47.63	11.61	5.30	22.94
3	8	49.10	12.01	5.48	22.94
3	9	50.33	12.63	5.80	22.94
4	1	23.96	2.53	1.66	11.34
4	2	36.70	6.45	3.42	17.42
4	3	43.33	9.68	4.45	23.85
4	4	44.91	11.16	5.36	24.83
4	5	47.08	11.91	5.73	24.83
4	6	49.10	13.66	6.28	24.83
4	7	50.52	14.20	6.52	25.81
4	8	51.04	14.74	6.68	25.81
4	9	51.36	15.36	7.01	25.81
5	1	26.76	3.88	2.39	13.61
5	2	39.89	8.33	4.10	19.19
5	3	44.32	11.65	5.43	25.96
5	4	47.08	12.97	6.22	26.75
5	5	49.47	13.84	6.58	26.75
5	6	50.94	15.52	7.08	26.75
5	7	51.31	16.09	7.34	27.53
5	8	53.30	16.58	7.49	27.53
5	9	54.34	17.11	7.76	27.53

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
6	1	32.92	4.92	2.91	15.34
6	2	43.33	9.87	4.73	21.73
6	3	46.16	13.02	6.11	27.37
6	4	49.10	14.40	6.84	28.02
6	5	50.94	15.40	7.20	28.03
6	6	51.36	16.92	7.68	28.03
6	7	53.83	17.42	7.91	28.68
6	8	54.57	17.87	8.07	28.68
6	9	56.46	18.38	8.33	28.68
7	1	35.30	6.24	3.35	17.51
7	2	44.14	11.25	5.39	23.54
7	3	47.63	14.16	6.64	28.38
7	4	50.52	15.62	7.33	28.94
7	5	51.31	16.61	7.70	28.94
7	6	53.83	17.94	8.12	28.94
7	7	54.66	18.44	8.38	29.50
7	8	56.57	18.87	8.52	29.50
7	9	57.61	19.54	8.95	29.50
8	1	36.70	7.33	3.73	19.62
8	2	44.91	12.32	5.90	24.90
8	3	49.10	15.14	7.08	29.13
8	4	51.04	16.63	7.74	29.62
8	5	53.30	17.54	8.08	29.63
8	6	54.57	18.77	8.51	29.63
8	7	56.57	19.25	8.74	30.11
8	8	58.02	19.83	9.05	30.12
8	9	58.39	20.48	9.46	30.12
9	1	38.16	8.28	4.07	21.26
9	2	46.16	13.20	6.32	25.96
9	3	50.33	16.01	7.44	29.72
9	4	51.36	17.44	8.07	30.16
9	5	54.34	18.30	8.41	30.16
9	6	56.46	19.45	8.81	30.16
9	7	57.61	20.06	9.18	30.59
9	8	58.39	20.66	9.49	30.59
9	9	59.53	21.27	9.87	30.59

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

Table 6. Adaptive Branch Predictor using 3-bit Counters

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
2	1	10.41	0.00	0.00	0.01
2	2	28.87	1.72	1.72	19.72
2	3	36.42	4.90	2.66	19.75
2	4	40.38	6.13	3.53	19.75
2	5	42.40	6.58	3.68	19.75
2	6	45.09	7.56	3.91	19.76
2	7	45.74	8.20	4.24	19.76
2	8	46.64	8.56	4.41	19.76
2	9	48.03	8.80	4.57	20.43
3	1	22.99	1.15	1.14	13.15
3	2	36.42	5.23	3.50	26.31
3	3	41.22	8.31	4.38	26.34
3	4	45.09	9.79	5.29	26.34
3	5	45.98	10.42	5.60	26.79
3	6	48.03	11.50	6.12	26.79
3	7	49.51	12.03	6.40	26.79
3	8	50.23	12.40	6.61	26.79
3	9	51.49	12.78	6.80	27.24
4	1	28.87	3.31	2.19	19.73
4	2	40.38	7.70	4.58	29.61
4	3	45.09	10.63	5.57	29.97
4	4	46.64	12.19	6.60	29.97
4	5	48.93	12.77	6.90	30.31
4	6	50.23	13.81	7.40	30.31
4	7	51.68	14.42	7.69	30.32
4	8	52.36	14.79	7.88	30.32
4	9	52.96	15.23	8.10	30.65
5	1	31.96	5.10	3.16	23.69
5	2	42.40	9.59	5.43	31.59
5	3	45.98	12.38	6.56	32.15
5	4	48.93	13.77	7.48	32.15
5	5	50.75	14.47	7.80	32.42
5	6	52.24	15.44	8.26	32.43
5	7	52.87	16.07	8.56	32.43
5	8	54.53	16.44	8.74	32.43
5	9	55.57	16.85	8.93	32.70

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Percent of Total (Average)			
		Short Penalty	Potential Savings	Measured Savings	Best Case
6	1	36.42	6.44	3.86	26.32
6	2	45.09	11.01	6.10	33.14
6	3	48.03	13.62	7.27	33.60
6	4	50.23	15.01	8.13	33.60
6	5	52.24	15.76	8.46	33.83
6	6	52.96	16.69	8.91	33.84
6	7	55.35	17.27	9.17	33.84
6	8	55.92	17.65	9.38	33.84
6	9	58.82	18.07	9.60	34.07
7	1	38.98	7.68	4.43	28.21
7	2	45.74	12.20	6.73	34.24
7	3	49.51	14.63	7.84	34.64
7	4	51.68	16.01	8.64	34.65
7	5	52.87	16.79	8.99	34.84
7	6	55.35	17.64	9.39	34.85
7	7	56.04	18.24	9.71	34.85
7	8	58.93	18.61	9.90	34.85
7	9	60.00	19.28	10.34	35.06
8	1	40.38	8.77	4.93	29.62
8	2	46.64	13.14	7.23	35.07
8	3	50.23	15.50	8.30	35.42
8	4	52.36	16.84	9.06	35.42
8	5	54.53	17.60	9.40	35.60
8	6	55.92	18.45	9.83	35.60
8	7	58.93	19.01	10.12	35.61
8	8	60.68	19.61	10.51	35.62
8	9	61.01	20.26	10.93	35.80
9	1	41.22	9.70	5.37	30.72
9	2	48.03	13.91	7.65	35.71
9	3	51.49	16.22	8.68	36.03
9	4	52.96	17.53	9.41	36.03
9	5	55.57	18.28	9.75	36.19
9	6	58.82	19.10	10.18	36.19
9	7	60.00	19.86	10.64	36.20
9	8	61.01	20.47	11.02	36.21
9	9	62.10	21.07	11.41	36.37

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

5. Implementation schemes

There are many ways to prevent the unnecessary flushing of instructions already in the pipeline. Perhaps the simplest is to maintain a stack of the addresses that have been fetched and have not yet been issued. Then, when a branch condition is evaluated, the stack can be searched associatively for the target address - if it is found, then the processor's *existing* instruction nullification logic can be used to nullify all the instructions between the branch and the branch target. All remaining instructions in the pipeline are allowed to continue. Since many newer pipeline organizations delay register operand reads until relatively late in the pipeline, there should be few, if any, invalid register reads. Stated another way, until the data for an instruction is fetched, there is no need to determine if invalid data has been forwarded from those incorrectly executed instruction to the ones allowed to continue. Our scheme is attractive because it makes use of the existing nullification logic, and only requires a single extra structure to be added (the stack of addresses).

This scheme can be thought of as providing functionality very similar to that of predicated execution. In a machine that supports predicated execution, control flow operations (branches) are replaced by predicates which control the completion of instructions. In order to accomplish this, the compiler performs a predicate transformation which statically replaces branch instructions with instructions that will issue unconditionally and complete conditionally. Since the branches have been removed, there can be no branch penalty. However, each predicated instruction must be issued, so processor resources are tied up by instructions that will not be allowed to complete. A trade-off exists between branch penalty (C*W) and instruction count.

Our modified scheme of misprediction recovery pays the same penalty as if we had performed a predicate transformation of the branch when we incorrectly predict the branch to be not taken. Unlike predicate transformation, branch prediction still takes place and performance will not suffer when the branch is Taken (when we correctly predict that condition). We therefore achieve some of the benefit of predicate transformation without requiring architecturally visible conditional predicate registers; the predicates are determined dynamically in the stack of addresses.

6. Hybrid Predictors

We have already observed that this technique can only reduce the branch penalties of short forward branches that are incorrectly predicted to be not taken. Therefore, we decided to look at what would happen if we biased a predictor towards predicting more "not taken" events. Such an approach will likely increase the number of incorrectly predicted branches, but could potentially decrease the overall branch penalty since managed flushing of the pipeline will mitigate the impact of these incorrectly predicted branches.

For example, for a 2-bit counter, the four prediction states are normally *Strongly Not Taken* (00), *Weakly Not Taken* (01), *Weakly Taken* (10) and *Strongly Taken* (11). In a new set of simulations we changed the interpretation (for short forward branches only) to be *Strongly Not Taken* (00), *Moderately Not Taken* (01), *Weakly Not Taken* (10) and *Strongly Taken* (11). This effectively skews the prediction towards Not Taken while requiring no modification to the normal translations determined from the calculated branch outcome. Essentially, all that happens is that the meaning of the bit pattern is interpreted differently for short forward branches. Entries corresponding to short forward branches operate in the same manner as those outside that range except in assigning a prediction direction. Therefore, there is no need for a separate table to handle the short forward branches.

Tables 7 and 8 contain measurements of the accuracy of the different branch predictors and the various counter states for both short forward branches and for all branches. In these tables we can see that for all schemes, approximately 26% of the time the predictor is in the *Strongly Not Taken* state, about 66% of the time it is in the *Strongly Taken* state, and that both of these states have accuracies in the 90's. However, the accuracy of the intermediate states (the *Weak* states) is dramatically lower (as low as 46.85%). Since the prediction accuracy of these states is so poor, there is reason to believe that changing their predictions to be *Not Taken* instead of *Taken* will not dramatically increase the miss rate, and will allow us to apply our managed pipeline flushing technique to actually lower the overall branch penalty.

Tables 9 and 10 present the results of simulations using these skewed predictors, calculating the Potential savings in the same manner as it was calculated in tables 3-6. In each table, C and W also represent the same things they did in tables 3-6 (the pre-execute pipeline stage count and the issue width, respectively). The *No Shift* columns show the Potential savings calculated without any skewing of the predictors. The *n Shift* columns contain the Potential reduction in

Table 7. Branch Prediction Accuracy of the different states of the 2-bit Branch predictors

Range	**Prediction Scheme**		**Prediction State Bit Pattern**			
			00	01	10	11
All Instructions	Simple 2-bit	% of time in State	25.98	3.14	4.14	66.73
		Accuracy of State	91.20	66.08	78.70	94.69
	Adaptive 2-bit	% of time in State	27.58	1.92	2.57	67.94
		Accuracy of State	95.04	63.64	77.73	97.08
0-12 Instructions	Simple 2-bit	% of time in State	28.04	6.63	7.95	57.38
		Accuracy of State	87.17	55.51	63.67	91.17
	Adaptive 2-bit	% of time in State	31.13	4.07	5.19	59.61
		Accuracy of State	92.31	55.48	66.12	94.47

Table 8. Branch Prediction Accuracy of the different states of the 3-bit Branch predictors

Range	Prediction Scheme		Prediction State Bit Pattern							
			000	001	010	011	100	101	110	111
All Instructions	Simple 3-bit	% of time in State	25.01	2.24	0.88	0.75	0.81	1.06	3.44	65.82
		Accuracy of State	92.68	73.43	56.42	48.78	54.37	60.43	83.96	95.10
	Adaptive 3-bit	% of time in State	27.03	1.32	0.63	0.51	0.51	0.67	1.99	67.33
		Accuracy of State	95.89	73.87	58.53	49.78	54.54	65.82	84.25	97.44
0-12 Instructions	Simple 3-bit	% of time in State	25.39	3.04	1.77	2.06	2.59	3.75	6.24	55.17
		Accuracy of State	91.22	72.08	48.31	46.85	55.95	60.30	66.87	92.40
	Adaptive 3-bit	% of time in State	29.40	2.40	1.62	1.47	1.53	1.88	3.78	57.92
		Accuracy of State	94.22	64.15	54.51	49.49	52.29	58.70	73.16	95.41

branch penalty when the predictor is skewed by *n* states. So, for example, in the *3 Shift* column of Table 10, intermediate states 100, 101 and 110 (normally weak Taken states) have been interpreted as weak Not Taken states instead. For each prediction scheme, the bold numbers indicate the largest savings for that product of C and W.

The results for the skewed 2-bit counters (Table 9) indicate that until the product of C and W gets large enough, the skewing provides little benefit. However, Table 10 shows that in the case of the 3-bit counters, skewing the predictor starts paying dividends almost immediately, and skewing it two places (moving 100 and 101 from the "predict taken" to the "predict not taken" camps) almost invariably increases the potential savings. And in many cases the increase in Potential branch reduction is quite noteworthy (from 15% to 20%, for example).

The reduction in branch penalty comes because the branch penalty is now a function of branch distance, not just of prediction accuracy. In almost all cases shown in these figures the actual number of mispredicted branches increased as we skewed the predictor, but because the branch penalty calculation is no longer symmetric (both misprediction types having equal penalty) an increase in misprediction rate does not necessarily translate into increased execution time.

The more the predictor is skewed, the closer we come to the permanent predict NOT TAKEN state. This state provides the same performance as performing predicate transformation (without rescheduling). However, we have previously shown that when the predictor is in a strongly taken state, the branch will be taken more than 95% of the time (Tables 7 and 8). By using skewed predictors, we essentially provide a less aggressive alternative to predicate transformation.

Table 9. Potential Savings using Modified 2-bit Counter

C	W	Simple 2-bit Predictor		Adaptive 2-bit Predictor	
		No Shift	1 Shift	No Shift	1 Shift
2	1	**0.00**	-2.40	**0.00**	-2.60
2	2	**1.37**	-1.79	**1.79**	-4.28
2	3	**4.27**	1.34	**4.90**	0.94
2	4	**5.22**	2.60	**6.14**	2.40
2	5	**5.65**	2.89	**6.59**	2.12
2	6	**7.65**	5.54	**7.55**	3.15
2	7	**8.17**	6.34	**8.20**	3.91
2	8	**8.57**	6.82	**8.56**	4.06
2	9	**9.12**	7.51	**8.79**	4.16
3	1	**0.92**	-1.57	**1.19**	-4.28
3	2	**4.40**	2.20	**5.29**	1.62
3	3	**7.94**	7.03	**8.30**	6.44
3	4	**9.19**	8.65	**9.80**	7.62
3	5	10.07	**10.10**	**10.43**	8.48
3	6	12.02	**12.62**	**11.48**	9.66
3	7	12.48	**13.45**	**12.02**	10.57
3	8	12.86	**13.69**	**12.40**	10.99
3	9	13.49	**14.37**	**12.78**	10.98
4	1	**2.82**	0.52	**3.35**	-1.11
4	2	**7.12**	6.09	**7.74**	5.53
4	3	10.52	**10.62**	**10.62**	9.27
4	4	12.09	**13.11**	**12.18**	11.39
4	5	12.84	**14.52**	**12.76**	12.30
4	6	14.58	**16.48**	**13.80**	13.79
4	7	15.15	**17.31**	**14.40**	**14.14**
4	8	15.69	**18.06**	14.78	**14.87**
4	9	16.31	**18.87**	15.23	**15.72**
5	1	**4.34**	2.45	**5.13**	2.09
5	2	**9.12**	8.72	**9.62**	7.99
5	3	12.57	**13.80**	**12.36**	12.17
5	4	13.96	**16.16**	13.76	**14.19**
5	5	14.83	**17.27**	14.46	**14.62**
5	6	16.50	**19.39**	15.43	**16.17**
5	7	17.09	**20.37**	16.06	**17.33**
5	8	17.58	**20.61**	16.42	**17.95**
5	9	18.11	**21.21**	16.85	**17.01**

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Simple 2-bit Predictor		Adaptive 2-bit Predictor	
		No Shift	1 Shift	No Shift	1 Shift
6	1	**5.50**	4.11	**6.48**	4.02
6	2	10.75	**11.48**	**11.03**	10.10
6	3	14.00	**16.15**	13.60	**13.92**
6	4	15.43	**18.17**	15.01	**16.20**
6	5	16.42	**19.64**	15.75	**16.95**
6	6	17.93	**21.66**	16.68	**18.63**
6	7	18.46	**21.97**	17.25	**19.46**
6	8	19.01	**22.78**	17.64	**18.50**
6	9	19.51	**23.69**	18.06	**19.64**
7	1	**6.90**	5.89	**7.71**	5.93
7	2	12.19	**13.58**	**12.22**	12.04
7	3	15.18	**18.01**	14.62	**15.85**
7	4	16.68	**19.99**	16.00	**17.43**
7	5	17.66	**21.56**	16.78	**18.95**
7	6	18.99	**22.78**	17.63	**20.26**
7	7	19.59	**23.75**	18.23	**19.58**
7	8	20.09	**24.80**	18.59	**20.65**
7	9	20.78	**25.59**	19.24	**21.83**
8	1	**8.08**	7.78	**8.80**	7.60
8	2	13.30	**15.31**	13.15	**13.35**
8	3	16.18	**19.27**	15.49	**17.13**
8	4	17.70	**21.56**	16.83	**18.99**
8	5	18.61	**22.57**	17.58	**20.36**
8	6	19.91	**24.24**	18.43	**20.07**
8	7	20.47	**25.44**	18.99	**21.36**
8	8	21.14	**26.27**	19.56	**22.35**
8	9	21.80	**27.12**	20.19	**23.52**
9	1	9.09	**9.47**	**9.72**	9.14
9	2	14.20	**16.79**	13.92	**14.60**
9	3	17.07	**20.57**	16.21	**17.85**
9	4	18.53	**22.84**	17.52	**20.31**
9	5	19.45	**23.80**	18.26	**19.94**
9	6	20.66	**25.68**	19.09	**21.58**
9	7	21.36	**26.71**	19.81	**22.87**
9	8	22.03	**27.63**	20.41	**23.93**
9	9	22.65	**28.33**	20.99	**24.85**

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

Table 10. Potential Savings using Modified 3-bit Counter

C	W	Simple 3-bit Predictor				Adaptive 3-bit Predictor			
		No Shift	1 Shift	2 Shift	3 Shift	No Shift	1 Shift	2 Shift	3 Shift
2	1	**0.00**	-0.07	-0.47	-2.96	**0.00**	-0.12	-0.46	-3.23
2	2	**1.20**	1.12	0.58	-2.78	1.72	**1.88**	1.40	-5.41
2	3	3.85	4.32	**4.36**	0.70	4.90	**5.49**	5.44	0.05
2	4	4.65	5.27	**5.47**	1.82	6.13	6.86	**6.95**	1.56
2	5	5.07	5.75	**6.02**	2.14	6.58	7.35	**7.45**	1.22
2	6	7.06	7.84	**8.25**	4.84	7.56	8.42	**8.65**	2.35
2	7	7.50	8.36	**8.92**	5.76	8.20	9.15	**9.49**	3.22
2	8	7.92	8.81	**9.40**	6.29	8.56	9.54	**9.87**	3.33
2	9	8.44	9.37	**9.98**	6.89	8.80	9.79	**10.07**	3.34
3	1	**0.80**	0.74	0.24	-2.40	1.15	**1.31**	0.92	-5.11
3	2	3.90	4.41	**4.57**	1.65	5.23	5.90	**5.98**	0.88
3	3	7.28	8.24	**8.92**	6.68	8.31	9.32	**9.80**	6.02
3	4	8.37	9.50	**10.45**	8.44	9.79	10.98	**11.67**	7.35
3	5	9.24	10.46	**11.55**	10.02	10.42	11.67	**12.44**	8.28
3	6	11.22	12.65	**13.96**	12.81	11.50	12.85	**13.67**	9.48
3	7	11.61	13.13	**14.56**	13.71	12.03	13.45	**14.35**	10.39
3	8	12.01	13.56	**15.04**	14.02	12.40	13.87	**14.82**	10.84
3	9	12.63	14.23	**15.77**	14.73	12.78	14.31	**15.28**	10.82
4	1	2.53	**2.72**	2.54	-0.12	3.31	**3.69**	3.50	-1.90
4	2	6.45	7.33	**7.93**	5.76	7.70	8.66	**9.09**	5.08
4	3	9.68	10.94	**12.09**	10.64	10.63	11.92	**12.75**	9.16
4	4	11.16	12.69	**14.19**	13.50	12.19	13.66	**14.73**	11.51
4	5	11.91	13.54	**15.15**	14.95	12.77	14.30	**15.37**	12.35
4	6	13.66	15.47	**17.30**	17.20	13.81	15.46	**16.69**	13.96
4	7	14.20	16.08	18.02	**18.11**	14.42	16.16	**17.44**	14.35
4	8	14.74	16.67	18.65	**18.92**	14.79	16.58	**17.91**	15.11
4	9	15.36	17.36	19.41	**19.81**	15.23	17.06	**18.46**	16.01
5	1	3.88	4.37	**4.55**	2.06	5.10	5.75	**5.89**	1.55
5	2	8.33	9.48	**10.46**	8.67	9.59	10.79	**11.52**	7.83
5	3	11.65	13.21	**14.76**	14.26	12.38	13.89	**15.03**	12.40
5	4	12.97	14.78	16.63	**16.87**	13.77	15.44	**16.71**	14.48
5	5	13.84	15.73	17.70	**18.08**	14.47	16.20	**17.54**	14.89
5	6	15.52	17.58	19.74	**20.45**	15.44	17.31	**18.76**	16.56
5	7	16.09	18.24	20.51	**21.56**	16.07	18.02	**19.58**	17.83
5	8	16.58	18.45	20.77	**21.48**	16.44	18.43	**20.04**	18.49
5	9	17.11	19.04	21.41	**22.13**	16.85	18.88	**20.54**	17.55

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

C	W	Simple 3-bit Predictor				Adaptive 3-bit Predictor			
		No Shift	1 Shift	2 Shift	3 Shift	No Shift	1 Shift	2 Shift	3 Shift
6	1	4.92	5.65	**6.10**	3.86	6.44	7.27	**7.58**	3.54
6	2	9.87	11.19	**12.44**	11.59	11.01	12.36	**13.31**	10.12
6	3	13.02	14.79	16.61	**16.82**	13.62	15.26	**16.50**	14.22
6	4	14.40	16.40	18.50	**19.16**	15.01	16.83	**18.31**	16.66
6	5	15.40	17.48	19.71	**20.73**	15.76	17.67	**19.19**	17.41
6	6	16.92	19.17	21.58	**23.01**	16.69	18.72	20.40	**19.26**
6	7	17.42	19.43	21.93	**23.04**	17.27	19.36	**21.10**	20.14
6	8	17.87	19.98	22.57	**23.91**	17.65	19.78	**21.59**	19.22
6	9	18.38	20.54	23.19	**24.64**	18.07	20.30	**22.21**	20.28
7	1	6.24	7.12	**7.75**	5.64	7.68	8.64	**9.11**	5.58
7	2	11.25	12.77	**14.30**	13.99	12.20	13.70	**14.85**	12.29
7	3	14.16	16.11	18.15	**18.91**	14.63	16.40	**17.82**	16.26
7	4	15.62	17.77	20.08	**21.17**	16.01	17.97	**19.58**	18.01
7	5	16.61	18.86	21.29	**22.88**	16.79	18.83	**20.54**	19.62
7	6	17.94	20.02	22.60	**23.94**	17.64	19.78	**21.60**	21.02
7	7	18.44	20.64	23.36	**25.02**	18.24	20.45	**22.36**	20.42
7	8	18.87	21.16	23.98	**25.91**	18.61	20.91	**22.92**	21.41
7	9	19.54	21.89	24.78	26**.75**	19.28	21.64	**23.71**	22.66
8	1	7.33	8.39	**9.25**	7.71	8.77	9.87	**10.51**	7.39
8	2	12.32	14.00	15.74	**15.95**	13.14	14.74	**16.01**	13.70
8	3	15.14	17.21	19.42	**20.37**	15.50	17.38	**18.94**	17.69
8	4	16.63	18.90	21.38	**22.94**	16.84	18.90	**20.64**	19.70
8	5	17.54	19.60	22.18	**23.71**	17.60	19.74	**21.58**	21.16
8	6	18.77	21.00	23.77	**25.57**	18.45	20.68	**22.64**	20.97
8	7	19.25	21.60	24.50	**26.65**	19.01	21.37	**23.46**	22.22
8	8	19.83	22.27	25.26	**27.51**	19.61	22.00	**24.12**	23.22
8	9	20.48	22.98	26.03	**28.43**	20.26	22.71	**24.91**	24.48
9	1	8.28	9.46	**10.50**	9.46	9.70	10.92	**11.72**	9.09
9	2	13.20	15.01	16.90	**17.54**	13.91	15.60	**16.93**	14.99
9	3	16.01	18.18	20.55	**21.82**	16.22	18.21	**19.85**	18.48
9	4	17.44	19.82	22.43	**24.37**	17.53	19.67	**21.52**	21.14
9	5	18.30	20.49	23.23	**25.08**	18.28	20.50	**22.44**	20.81
9	6	19.45	21.81	24.72	**26.92**	19.10	21.47	**23.58**	22.47
9	7	20.06	22.54	25.60	**28.06**	19.86	22.30	**24.49**	23.85
9	8	20.66	23.22	26.36	**29.02**	20.47	22.97	**25.22**	24.95
9	9	21.27	23.87	27.05	**29.76**	21.07	23.61	**25.92**	25.92

C=# of Pre-issue Pipeline Stages, **W**=Issue Width

7. Conclusions

In this paper we have presented a new strategy for reducing the branch penalty due to short forward branches that are incorrectly predicted to be not taken. Currently, when a processor detects that a branch has been incorrectly predicted, all instructions in the pipeline past the branch are flushed. However, when the branch is a short forward branch and the misprediction was Not Taken, many of the desired target instructions have already been fetched and are in the pipeline. Flushing these instructions and then immediately refetching them is obviously inefficient, and as both pipeline depths and issue widths increase, this problem will become more and more onerous.

Our measurements of the programs in the SPEC benchmark suite indicate that short forward branches are not only common, but are the most difficult for branch prediction schemes to predict correctly. This explains why incorrectly predicted short forward branches account for a substantial amount of the total branch penalty.

We have proposed a two-pronged approach to reducing this penalty - intelligent management of the flushing process, and biasing the branch predictors for short forward branches to increase our ability to reduce the total branch penalty. By only flushing the instructions that are between the branch instruction and the target address, the branch penalty changes from a function of the number of mispredictions to a function of the branch distance. This is an important difference as pipeline depths and issue widths increase. It also supports the use of biased predictors, in order to capitalize on the weakness of branch prediction strategies to correctly predict short forward branches.

Simulations of this approach indicate that by implementing the managed flushing technique as much as 30% of the total branch penalty can be eliminated. The hardware modifications required to incorporate this new strategy make use of existing nullification logic, and should be easily incorporated into existing architectures. In addition, incorporating the biased predictors increases the effectiveness significantly and also requires minimal hardware modifications.

8. References

[BaGS94] D. F. Bacon, S. L. Graham and O. J. Sharp, ‘‘Compiler Transformations for High-Performance Computing’’, *ACM Computing Surveys*, vol. 26, no. 4 (December 1994), pp. 345-420.

[FiFr92] J. A. Fisher and S. M. Freudenberger, ‘‘Predicting Conditional Branch Directions from Previous Runs of a Program’’, *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA (October 12-15, 1992), pp. 85-95.

[LeS84] J. K. L. Lee and A. J. Smith, ‘‘Branch Prediction Strategies and Branch Target Buffer Design’’, *Computer*, vol. 17, no. 1 (January 1984), pp. 6-22.

[PaS92] S. Pan, K. So and J. T. Rahmeh, ‘‘Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation’’, *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA (October 12-15, 1992), pp. 76-84.

[Smit81] J. E. Smith, ‘‘A Study of Branch Prediction Strategies’’, *Proceedings of the Eighth Annual International Symposium on Computer Architecture*, Minneapolis, Minnesota (May 1981), pp. 135-148.

[SrWa94] A. Srivastava and D. W. Wall, ‘‘Atom: A system for building custonized program analysis tools’’, *Proceedings of the ACM SIGPLAN Notices 1994 Conference on Programming Languages and Implementations*(June 1994), pp. 196-205.

[YeP91] T. Yeh and Y. Patt, ‘‘Two-Level Adaptive Training Branch Prediction’’, *Proceedings of the 24th Annual International Symposium on Microarchitecture*, Albuquerque, New Mexico (November 18-20, 1991), pp. 51-61.

[YeP93] T. Yeh and Y. Patt, ‘‘A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History’’, *Proceedings of the Twentieth Annual International Symposium on Computer Architecture*, San Diego, CA (May 16-19, 1993), pp. 257-266.