Is branch prediction important for performance?

Daniel J. Bernstein

Spectre paper: “Modern processors use branch prediction and speculative execution to maximize performance.”

Wikipedia: “Branch predictors play a critical role in achieving high effective performance in many modern pipelined microprocessor architectures such as x86.”

The article cited by Wikipedia says: "Branch predictor (BP) is an essential component in modern processors since high BP accuracy can improve performance and reduce energy by decreasing the number of instructions executed on wrong-path."

The article cited by Wikipedia says: "Branch predictor (BP) is an essential component in modern processors since high BP accuracy can improve performance and reduce energy by decreasing the number of instructions executed on wrong-path."

— Omitting branch prediction reduces energy even more. Eliminates all wrong-path instructions. Also eliminates cost of prediction+speculation.

The article cited by Wikipedia says: "Branch predictor (BP) is an essential component in modern processors since high BP accuracy can improve performance and reduce energy by decreasing the number of instructions executed on wrong-path."

— Omitting branch prediction reduces energy even more. Eliminates all wrong-path instructions. Also eliminates cost of prediction+speculation.

The real question is **latency**.

The CPU pipeline

Cycle 1:

fetch	`a=b+c`
decode	
register read	
execute	
register write	

The CPU pipeline

Cycle 2:

fetch	
decode	`a=b+c`
register read	
execute	
register write	

The CPU pipeline

Cycle 3:

fetch	
decode	
register read	`a=b+c`
execute	
register write	

The CPU pipeline

Cycle 4:

fetch	
decode	
register read	
execute	`a=b+c`
register write	

The CPU pipeline

Cycle 5:

fetch	
decode	
register read	
execute	
register write	`a=b+c`

1 instruction finishes in 5 cycles.

The CPU pipeline

Another program, cycle 1:

fetch	`a=b+c`
decode	
register read	
execute	
register write	

The CPU pipeline

Cycle 2:

fetch	`d=e+f`
decode	`a=b+c`
register read	
execute	
register write	

Second instruction is fetched;
first instruction is decoded.
Hardware units operate in parallel.

The CPU pipeline

Cycle 3:

fetch	`g=h-i`
decode	`d=e+f`
register read	`a=b+c`
execute	
register write	

Third instruction is fetched;
second instruction is decoded;
first instruction does register read.

The CPU pipeline

Cycle 4:

fetch	`j=k+l`
decode	`g=h-i`
register read	`d=e+f`
execute	`a=b+c`
register write	

The CPU pipeline

Cycle 5:

fetch	`m=n-o`
decode	`j=k+l`
register read	`g=h-i`
execute	`d=e+f`
register write	`a=b+c`

Program continues this way.

Throughput: 1 instruction/cycle.

The CPU pipeline

Another program, cycle 1:

fetch	`a=b+c`
decode	
register read	
execute	
register write	

The CPU pipeline

Cycle 2:

fetch	`d=a-e`
decode	`a=b+c`
register read	
execute	
register write	

The CPU pipeline

Cycle 3:

fetch	`...`
decode	`d=a-e`
register read	`a=b+c`
execute	
register write	

The CPU pipeline

Cycle 4:

fetch	...
decode	...
register read	`d=a-e`
execute	`a=b+c`
register write	

Register-read unit is idle,
waiting for a to be ready.

The CPU pipeline

Cycle 5:

fetch	...
decode	...
register read	`d=a-e`
execute	
register write	`a=b+c`

Execute unit is idle.

Typical CPUs design pipelines
to eliminate this slowdown:
fast-forward a to next operation.

The CPU pipeline

Another program, cycle 1:

fetch	`a=b+c`
decode	
register read	
execute	
register write	

The CPU pipeline

Cycle 2:

fetch	`d=e+f`
decode	`a=b+c`
register read	
execute	
register write	

The CPU pipeline

Cycle 3:

fetch	`g=h-i`
decode	`d=e+f`
register read	`a=b+c`
execute	
register write	

The CPU pipeline

Cycle 4:

fetch	`if(g<0)`
decode	`g=h-i`
register read	`d=e+f`
execute	`a=b+c`
register write	

The CPU pipeline

Cycle 5:

fetch	
decode	`if(g<0)`
register read	`g=h-i`
execute	`d=e+f`
register write	`a=b+c`

Without branch prediction,
fetch unit doesn't know
which instruction to fetch now!
Waiting for `if` to write
"instruction pointer" register.

The CPU pipeline

Cycle 6:

fetch	
decode	
register read	`if(g<0)`
execute	`g=h-i`
register write	`d=e+f`

Fetch is still waiting.

Typical CPUs: longer pipelines; longer delays than this picture. (Assume no hyperthreading.)

The CPU pipeline

Cycle 5, speculative execution:

fetch	`g=-g`
decode	`if(g<0)`
register read	`g=h-i`
execute	`d=e+f`
register write	`a=b+c`

Branch predictor guesses
which instruction to fetch.
More work to undo everything
if guess turns out to be wrong,
but usually guess is correct.

The CPU pipeline

Better program, cycle 1:

fetch	`<0? g=h-i`
decode	
register read	
execute	
register write	

The CPU pipeline

Cycle 2:

fetch	`a=b+c`
decode	`<0? g=h-i`
register read	
execute	
register write	

The CPU pipeline

Cycle 3:

fetch		`d=e+f`
decode		`a=b+c`
register read	`<0?`	`g=h-i`
execute		
register write		

The CPU pipeline

Cycle 4:

fetch		`j=k+l`
decode		`d=e+f`
register read		`a=b+c`
execute	`<0?`	`g=h-i`
register write		

The CPU pipeline

Cycle 5:

fetch	`if(?)`
decode	`j=k+l`
register read	`d=e+f`
execute	`a=b+c`
register write	`<0? g=h-i`

Fast-forward flag to fetch unit.

Branch prediction has zero benefit **if programs compute branch conditions P cycles in advance**, where P is pipeline length.

CPUs today spend almost all time applying simple computations to large volumes of data. Massively parallelizable.

Why shouldn't programs compute branch conditions in advance?

CPUs today spend almost all time
applying simple computations
to large volumes of data.
Massively parallelizable.

Why shouldn't programs compute
branch conditions in advance?

Most cases are handled by
simple instruction scheduling.

CPUs today spend almost all time
applying simple computations
to large volumes of data.
Massively parallelizable.

Why shouldn't programs compute
branch conditions in advance?

Most cases are handled by
simple instruction scheduling.

Insn-set extensions for more cases:
"branch-relevant" priority bit;
multiple flags; loop counter.
(Count down early in pipeline.)
Inner loops I've studied don't
need more complicated patterns.

How did the community convince itself that branch prediction is important for performance?

How did the community convince itself that branch prediction is important for performance?

1980s insn sets, CPU costs $\rightarrow$ 1990s compilers, applications, data volumes, compiled code $\rightarrow$ 1990s/2000s hype (e.g., "Since programs typically encounter branches every 4–6 instructions, inaccurate branch prediction causes a severe performance degradation in highly superscalar or deeply pipelined designs") $\rightarrow$ 2000s/2010s beliefs.

The fundamental question:
Can a well-designed insn set
with well-designed software
remove the speed incentive
for branch prediction?

The fundamental question:
Can a well-designed insn set
with well-designed software
remove the speed incentive
for branch prediction?

"We need to look at
current insn sets."

The fundamental question:
Can a well-designed insn set
with well-designed software
remove the speed incentive
for branch prediction?

"We need to look at
current insn sets." — Yes,
interesting short-term question.
Not my question in this talk.

The fundamental question:
Can a well-designed insn set
with well-designed software
remove the speed incentive
for branch prediction?

"We need to look at
current insn sets." — Yes,
interesting short-term question.
Not my question in this talk.

"We need to look at
badly written software."

The fundamental question:
Can a well-designed insn set
with well-designed software
remove the speed incentive
for branch prediction?

"We need to look at
current insn sets." — Yes,
interesting short-term question.
Not my question in this talk.

"We need to look at
badly written software." — No.
Obsolete view of performance.
Need well-designed software
for good speed already today.

"Fundamentally, you *cannot* compute branches in advance for these important computations. Look at, e.g., `int32[n]` heapsort. Inspect data, branch, repeat."

"Fundamentally, you *cannot*
compute branches in advance
for these important computations.
Look at, e.g., `int32[n]` heapsort.
Inspect data, branch, repeat."

— The current speed records for
`int32[n]` sorting on Intel CPUs
are held by sorting networks!
Data-independent branches
defined purely by `n`. Performance,
parallelizability, predictability
have clear connections.

`sorting.cr.yp.to`:
software + verification tools.