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.”

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."

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

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."

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

The real question is **latency**.

n prediction
nt for performance?

. Bernstein

---

paper: "Modern
ors use branch prediction
culative execution to
e performance."

ia: "Branch predictors
ritical role in achieving
ective performance
modern pipelined
ocessor architectures
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."

— 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

Cycle 1:

fetch
decode
register
execute
register

on
ormance?

n

Modern
nch prediction
xecution to
ance."

ch predictors
in achieving
ormance
ipelined
chitectures

1

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**.

2

The CPU pipeline

Cycle 1:

fetch
decode
register read
execute
register write

ction

ors

ng

1

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**.

2

The CPU pipeline

Cycle 1:

fetch
decode
register read
execute
register write

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 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 2:

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

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 3:

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

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 4:

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

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 5:

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

1 instruction finishes in 5 cycles.

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

Another program, cycle 1:

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

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 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 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 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 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 4:

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

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 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 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

Another program, cycle 1:

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

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 2:

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

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 3:

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

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 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 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 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 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

Another program, cycle 1:

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

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 2:

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

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 3:

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

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 4:

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

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 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 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 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 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 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 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

Better program, cycle 1:

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

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 2:

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

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 3:

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

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 4:

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

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 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.

cle cited by Wikipedia
Branch predictor (BP) is
tial component in modern
ors since high BP accuracy
rove performance and
energy by decreasing
ber of instructions
d on wrong-path."

ting branch prediction
energy even more.
es all wrong-path
ons. Also eliminates
prediction+speculation.

question is **latency**.

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 to
applying
to large
Massivel

Why sho
branch c

y Wikipedia
dictor (BP) is
onent in modern
gh BP accuracy
rmance and
decreasing
tructions
g-path."

ch prediction
en more.
ng-path
eliminates
+speculation.

is **latency**.

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
applying simple co
to large volumes o
Massively paralleli

Why shouldn't pro
branch conditions

lia
?) is
nodern
curacy
nd

on

s

on.

.

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 a
applying simple computation
to large volumes of data.
Massively parallelizable.

Why shouldn't programs co
branch conditions in advanc

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?

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?

Most cases are handled by
simple instruction scheduling.

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?

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.

U pipeline

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

ward flag to fetch unit.
prediction has zero benefit
ams compute branch
ons P cycles in advance,
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?

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
itself tha
importar

`if(?)`
`j=k+l`
`d=e+f`
`a=b+c`
`<0? g=h-i`

to fetch unit.
has zero benefit
pute branch
les in advance,
e 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?

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 comm
itself that branch
important for perf

`if(?)`
`j=k+l`
`d=e+f`
`a=b+c`
`g=h-i`

nit.

benefit

ch

ance,

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 con
itself that branch prediction
important for performance?

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?

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?

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.

oday spend almost all time
simple computations
volumes of data.
y parallelizable.

ouldn't programs compute
conditions in advance?

ses are handled by
nstruction scheduling.

extensions for more cases:
-relevant" priority bit;
flags; loop counter.
down early in pipeline.)
ops I've studied don't
ore complicated patterns.

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 fun
Can a w
with wel
remove
for bran

l almost all time
omputations
of data.
zable.

ograms compute
in advance?

ndled by
scheduling.

s for more cases:
priority bit;
p counter.
y in pipeline.)
udied don't
cated patterns.

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
Can a well-designe
with well-designed
remove the speed
for branch predicti

ll time
ns

mpute
e?

g.

cases:
t;

ne.)
't
erns.

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?

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?

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?

"We need to look at current insn sets."

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?

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

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?

"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."

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?

"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.

5

the community convince
at branch prediction is
nt for performance?

sn sets, CPU costs →
ompilers, applications,
umes, compiled code →
000s hype (e.g., “Since
s typically encounter
s every 4–6 instructions,
te branch prediction
severe performance
tion in highly superscalar
y pipelined designs”) →
010s beliefs.

6

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.

“Fundar
compute
for these
Look at,
Inspect

munity convince
prediction is
ormance?

PU costs →
pplications,
npiled code →
(e.g., “Since
encounter
6 instructions,
prediction
erformance
hly superscalar
d designs”) →
fs.

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, y
compute branches
for these importan
Look at, e.g., `int`
Inspect data, bran

vince
is
→
s,
e →
nce
r
ons,
calar
) →

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 advanc
for these important computa
Look at, e.g., `int32[n]` hea
Inspect data, branch, repeat

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."

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."

— 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.