depreciate JALR with low bit set. #625

Open David-Horner opened this issue on Feb 8, 2021 · 25 comments

David-Horner commented on Feb 8, 2021 Contributor

depreciate JALR with low bit set.

First make it reserved. Then determine its best use.

We could provide an optional compatibility mode. But it will not be used.
No compiler generates this in generated code, nor would it. It is useless.

A possible use is as the high bit below sign in the offset of a revised JALR
If the new code is run on an old implementation the problem should manifest in an obvious way.
To quote spec:

> Although there is potentially a slight loss of error checking in this case, in practice
> jumps to an incorrect instruction address will usually quickly raise an exception.

Note: No idea what TG or SIG or oversight group should consider this.
Note2: this suggestion is an outcome of the related topic raised in code-size-reduction TG
riscv/riscv-code-size-reduction#24 #issue-800076044
that is best considered at a higher level than that TG.

allenjbaum commented on Feb 10, 2021

I'd file an issue (or, if you're ambitious, a PR) in the spec and see what
happens.
I'd like an official word on what the intent of the way it was defined is.
…

David-Horner commented on Feb 10, 2021 Contributor Author

> On 2021-02-10 2:54 a.m., Allen Baum wrote:
> I'd file an issue (or, if you're ambitious, a PR) in the spec and see what happens.

Isn't this, riscv/riscv-isa-manual, the correct spec?

I'll consider a Pull Request, but

1) if this is not the correct spec, how will that get to the correct
people any better?

2) is a PR really better/more received/responded to than an issue? Why?
I expect a "FIX ME" patch would not be much appreciated.

    It appears to me that many immediately accepted PRs are discussed
"off-list" and worked out informally.

    Others are discussed extensively on the correct/appropriate list,
that is why I asked which forum that is for such an issue as this.

3) Issues/PRs of this type, that may not be considered just a "detail
clarification", InMyHumbleOpion need at least an explanatory note;

    Substantial discussion on detecting and mitigating the lack of
backward compatibility [however minimal] benefit from a deep
understanding before spec text changes.

> I'd like an official word on what the intent of the way it was defined is.

Agreed. But, we do have that in detail in isa-dev@groups.riscv.org back
in 2016. <
https://groups.google.com/a/groups.riscv.org/forum/?utm_source=digest&utm_medium=email#!forum/sw-dev/topics>

https://groups.google.com/a/groups.riscv.org/g/isa-dev/c/3nJnbTmrXTw/m/UZHzzSW2AgAJ
[isa-dev] JALR - potential hazard and suggestion that JALR with odd
immediate be NSE

https://groups.google.com/a/groups.riscv.org/g/isa-dev/c/6TICoQ3VLJo/m/gGpAudx1BwAJ

Test cases for SLT(I), SLTU(I)
The *JALR* with *odd* *immediate* is an anomaly that continues to be
problematic, so I again suggest it be a Non-Standard Extension. *JALR* -
potential *hazard* and *suggestion* that *JALR* with *odd*
<https://groups.google.com/a/groups.riscv.org/g/isa-dev/c/6TICoQ3VLJo/m/gGpAudx1BwAJ>

I don't have any other links than these, although discussion must have
occurred elsewhere.

I recall Krste and Andrew responding, saying too late [for the then
"reserved" meaning].
…

jrtc27 commented on Feb 16, 2021 Contributor

No compiler generates it, but language runtimes have been free to use it and could well be, and any change breaks backwards compatibility and risks breaking existing code written to a ratified spec.

allenjbaum commented on Feb 16, 2021

There have been non-backwards-compatible changes made if there was no backwards compatibly expected.
(see the ISA spec preface for a list).

So, I don't disagree - but we should clearly do some investigation to see if anyone is using (or even proposing to use) the low order bit in branch addresses before this goes any further.

How difficult is that at this point in RISC-Vs evolution?
Are the backwards compatibility issues in language runtimes not specific to RISC-V - that is, is it a technique that is currently used iin other architectures?

jrtc27 commented on Feb 16, 2021 Contributor

> There have been non-backwards-compatible changes made if there was no backwards compatibly expected.
> (see the ISA spec preface for a list).

Not since ratification of the standard extension in question (or in this case base ISA) that I'm aware of, just adding new instructions.

David-Horner commented on Feb 16, 2021 Contributor Author

@jrtc27
A most valid point.

There are limited methods to guard against the risk of breaking code.

The best appears to be the loader scanning for the low bit set in the jalr in code not known to be compiled with the future replacement functionality.

Language runtimes can, therefore, also checked for such code. All such mods would be hand-coded.

The current "feature" is brittle, rarely useful and easily replaced by conditional code, even by the loader in many use cases.

It is for these reason that I see it as a candidate for depreciating.

jrtc27 commented on Feb 16, 2021 Contributor

What "loader" is scanning JALRs, especially in a JITed context?

jrtc27 commented on Feb 16, 2021 Contributor

Also it's really not brittle, it's well defined and easy to understand IMO, JALR just calculates the address and masks of the low bit as a normal legalisation like RISC-V does all over the place.

allenjbaum commented on Feb 16, 2021

See the preface of the priv spec v1.12 (which is post-ratification), page
1, labelled
"The following changes have been made since version 1.11, which, while
not strictly backwards compatible, are not anticipated to cause software
portability problems in practice:"
Clearly, the kinds of changes here may not apply to runtimes; these are
all in corners of the priv spec, so our mileage will vary.
But, this was just pointing out that there is a precedent for non-backwards
compatible changs.
I think the burden is on us (for some value of "us" that doesn't include
me, ahem) to show that the low bit isn't used.
…

jnk0le commented on Feb 16, 2021 • edited

> we should clearly do some investigation to see if anyone is using (or even proposing to use) the low order bit in branch addresses before this goes any further.

There is one use case: emb-riscv/specs-markdown/issues/3
Requires that the function returns never use `jalr` with lsb set in the imm field as well as prohibits presence of "auxiliary information" in lsb of pointers passed to `jalr`.

David-Horner commented on Feb 16, 2021 Contributor Author

@jrtc27 > JALR just calculates the address and masks of the low bit as a normal legalisation like RISC-V does all over the place.

This calculation is unique among the RISCV instructions.
It is only in JALR that the address low bit is cleared.
All other jump and branch instructions cannot generate an odd address. [they can generate an odd package address that will trap without C extension]
Even exceptions were carefully crafted to not require clearing the low address bit.
There is not architecture low bit clearing for load/store addressing.
I call it brittle because

1. To the best of my knowledge, there is no test in the suite for this use case. [The actively maintained riscv/riscv-test omits it], I am looking through other test suites]. Hardware can easily be doing it wrongly and virtually no one would notice.
2. unintentionally triggering the feature occurs when two potentially unrelated sentinel bit approaches are used simultaneously.
   The feature will work reliably if it is intended and explicitly coded, but the conflation problem makes the use brittle.
   This combined with point 1, in which the hardware clears both bits before adding, makes a working program fail on a compliant implementation. [ed. of course, relying on a flawed implementation is "brittle".] The point is that this misimplementation will not likely show up in the vendors errata for the chip. It is of such low import to virtually everyone that due diligence cannot be expected for its detection and certainly not its reporting when the vendor has 1 million of the chip in stock.
   Adding the test case for verification/validation/certification does not really solve this problem, whereas depreciating does. It is a vendor friendly solution.

allenjbaum commented on Feb 16, 2021

Oops - if there is no test for it, you've found a hole in the test suite.
We will fix that (assuming this doesn't get much farther). Thank you.
...

jrtc27 commented on Feb 16, 2021 Contributor

> Oops - if there is no test for it, you've found a hole in the test suite. We will fix that (assuming this doesn't get much farther). Thank you.
> ...

Yeah, testing gaps for things like this are trivial to fix and not a reason to remove a minor quirk of the ISA...

David-Horner commented on Feb 16, 2021 Contributor Author

@jrtc27 >Requires that the function returns never use jalr with lsb set in the imm field as well as prohibits presence of "auxiliary information" in lsb of pointers passed to jalr.

So it is depreciated for user land in emb-riscv specs already.
It is worthwhile that there is no note in the issue to guard against inadvertent triggering of PC advancement with tagging jalr low bit for other purposes, such as debugger tracking/tracing state.
This adds to my argument that the combination is indeed brittle.

jrtc27 commented on Feb 16, 2021 Contributor

> **@jrtc27** >Requires that the function returns never use jalr with lsb set in the imm field as well as prohibits presence of "auxiliary information" in lsb of pointers passed to jalr.
>
> So it is depreciated for user land in emb-riscv specs already.
> It is worthwhile that there is no note in the issue to guard against inadvertent triggering of PC advancement with tagging jalr low bit for other purposes, such as debugger tracking/tracing state.
> This adds to my argument that the combination is indeed brittle.

emb-riscv is a random non-standard thing, not a ratified specification by the RISC-V Foundation. Being for the embedded space it also has very different concerns to a core running a general-purpose OS with thousands of pre-compiled binary packages.

I don't get what your point is. It's completely well-defined, and like anything if you use it in broken ways then the end-result is broken, but that doesn't mean it's not and cannot be used.

David-Horner commented on Feb 16, 2021 Contributor Author

My point is an ISA should be designed to avoid surprises.
RISCV accomplished this extremely well in my opinion.
It works in intuitive ways, when I see a feature and the way it is implemented, generally I say "of course".

I have worked in system programming with many other architectures.
IBM 360 [and 1130] had few surprises, their transition to 31bit addressing was somewhat convoluted, but readily understandable with few gotchas. IBM virtual mapping with segments was clunky.
x86 on the other hand, is terrible. Exposure to it makes one think that inane is the norm.

The issue is that two disparate characteristics combine to produce an unexpected result.
A on its own works, B on its own works, A + B goes bonkers, and the very debugging features used to check it out could be the cause of the debugging process failing. A land mine for anyone using A or B who decides to use both.

kasanovic commented on Feb 16, 2021 Collaborator

This would be a backwards-incompatible change in the unprivileged architecture. There is a much bigger barrier for unprivileged versus for the privileged architecture - basically it should be a major problem we're fixing and we haven't seen that in unprivileged since ratification. The privileged architecture changes Allen mentions mostly cleaned up ill-specified behavior, and changes are only visible to kernel code (a tiny fraction of all code, and code already designed to cope with architecture variations).

The instruction is perfectly well-specified and does not leak unexpected behavior around the rest to the system. There is no "surprise", beyond that this is not a particularly efficient use of encoding space. Making the low bit behave differently would add a new immediate encoding, which is its own form of surprise. I agree this use of low bit is not efficient, but does simplify implementations versus requiring different behavior (trap or new instruction).

As for actual uses, an admittedly contrived example is to allow different static call sites to select one of two entry points off of common function pointer, which dynamically either resolve to same entry point or two different entry points based on low bit of function pointer (call site code is unchanged, but flipping low bit of function pointer either combines or separates incoming calls to the two functions). For example, static call sites that output to either stream1 or stream2 determined by low bit of static offset, and have low bit of dynamic function pointer decide if these streams will be merged into one stream or separated depending on carryout of LSB. To be clear, this use was not motivation for original encoding design - simplicity was the reason for the encoding.

The lack of an architectural test should be fixed.

David-Horner commented on Feb 17, 2021 Contributor Author

> On 2021-02-16 5:40 p.m., Krste Asanovic wrote:
>
> This would be a backwards-incompatible change in the unprivileged
> architecture. There is a much bigger barrier for unprivileged versus
> for the privileged architecture - basically it should be a major
> problem we're fixing and we haven't seen that in unprivileged since
> ratification.

How major is major?

We haven't seen issues with this YET, possibly because we have not seen
the level of adoption of RISCV that would foster the wide spread use of

1) tagging jalr instructions - e.g. for debugging and tracing
purposes, and

2) tagging code target addresses as a sentinel value, directing
further behaviour.

As niche uses become common place for both of these "features" the
unintentional overlap causes unexpected behaviour.

Just as examining current compiler output to see what common instruction sequences can be aggregated into "optimizing instructions" primarily addresses current compiler inefficiencies,

So, too, asserting that a situation is not a concern because few have come forward with a complaint is short sighted.

It does not mean that problems are not occurring in the wild, only that no one has reported it yet.

It could be the same persons that are puzzled about this odd behaviour have just abandoned RISCV altogether because they believe it to be unreliable or too quirky.

> The privileged architecture changes Allen mentions mostly cleaned up ill-specified behavior, and changes are only visible to kernel code (a tiny fraction of all code, and code already designed to cope with architecture variations).
>
> The instruction is perfectly well-specified and does not leak unexpected behavior*around the rest to the system. *

? I suspect the sentence is manged somehow.

> There is no "surprise",

I disagree.

> beyond that this is not a particularly efficient use of encoding space.

But "not a particularly efficient use of encoding space" is not a surprise for those understanding the RISCV philosophy. Efficient coding was often sacrificed in RVI for summitry and straight forward hardware design.

You now start going over the discussions/debate of 2017

e.g.
https://groups.google.com/a/groups.riscv.org/g/isa-dev/c/3nJnbTmrXTw/m/UZHzzSW2AgAJ

> Making the low bit behave differently would add a new immediate encoding, which is its own form of surprise.

We survived the C extension with all its variants, the shift instructions are diverse in a comperable way [especially for RV128 which has a special shift encoding for 64!!I

I believe you over hype the emotional impact a this encoding variant will cause.

> I agree this use of low bit is not efficient, but does simplify implementations versus requiring different behavior (trap or new instruction).

Inefficiency is not my primary concern. Any benefits that depreciating odd jalr offsets could yield for reuse is an obvious plus, but secondary to avoiding future confusion and lots productivity; distress; embarrassment; loss of status/acceptance.

> As for actual uses, an admittedly contrived example is to allow different static call sites to select one of two entry points off of common function pointer, which dynamically either resolve to same entry point or two different entry points based on low bit of function pointer (call site code is unchanged, but flipping low bit of function pointer either combines or separates incoming calls to the two functions). For example, static call sites that output to either stream1 or stream2 determined by low bit of static offset, and have low bit of dynamic function pointer decide if these streams will be merged into one stream or separated depending on carryout of LSB.

Yes. I proposed similar "usefulness" of the feature back in 2017. Mine were as unconvincing as yours. The current "two odds makes a different even" jump is a virtually useless formulation.

The formulation "zero odds, one odd or the other odd, and both odd" behaves the same is much more understandable.

> To be clear, this use was not motivation for original encoding design - simplicity was the reason for the encoding.

And to be clear, I am not suggesting we highjack the encoding.

Back in 2017 the available alternative was to make the odd bit encoding a NSE, with an implicit trap.

With nuanced change in the meaning of "undefined", an alternative is possible.

What I am proposing is that odd offset JALR be depreciated. No one should rely on its currently stipulated behaviour.

As simple as that.

It provides guidance and a warning that you could be setting one half of
the trap if you use it. The other half set by those who set the low bit
of the branch address register.
…

allenjbaum commented on Feb 17, 2021

I just checked, and indeed it appears that, while we are testing for
unaligned JALR, and it does test for odd addresses - it does it
incorrectly, so base+offset are both odd.
Now i have to track down why this was missed in coverage rules, because it
should have been a pretty basic kind of test, ignoring the special case
aspect.
…

David-Horner commented on Feb 17, 2021 Contributor Author

@David-Horner

> warning that you could be setting one half of
> the trap if you use it. The other half set by those who set the low bit
> of the branch address register.

This is the stuff CVE are made from.
I don't doubt black hats have already logged this potential exposure. But if so why have they not reported it?

We have an opportunity to proactively avoid this exposure.
If we do not, I expect we will eventually be called out.

Is this not fodder for doctoral theses?
It is certainly material for a college course session in Architecture Design: Expedience vs Robustness.

I regret that I did not push aggressively back in 2017.
[so young and foolish back then, a different foolish now and feeling much older]
I will continue this crusade, I believe it is of substantial import.
In one way or another it must be publicized.
I believe depreciation by RISCV.org is the best means.
But other channels are available and should also be used.

allenjbaum commented on Feb 17, 2021

Aaaaand - I was just corrected. Those tests do generate odd jalr addresses.
We are still missing some coverage, but have at least 2 tests for that,

...

allenjbaum commented on Feb 17, 2021

I was mistaken (again). We do have tests that generate odd jalr addresses.
Not quite enough coverage there, but it is currently tested.

...

allenjbaum commented on Feb 17, 2021

I was just corrected.
While this isn't explicitly covered in the coverage tool, there are several
tests that generate odd target addresses and see that the bit is ignored.

On Tue, Feb 16, 2021 at 4:57 PM Allen Baum <allen.baum@esperantotech.com>
wrote:

...

David-Horner commented on Feb 17, 2021 Contributor Author

Yes, the even-even and odd-odd cases appear to be generated when I
looked at the

TEST_JALR_OP(tempreg, rd, rs1, imm, swreg, offset,adj) macro.

But I was happy to defer to the experts.

Offsetting adjustments were made to both immediate and register.

However, the ignoring of the single bit set case did not appear tested.

...

allenjbaum commented on Feb 17, 2021

The normal tests do have Base+odd offset. The fact that they don't trap and
end up jumping to the right place means that they are tested.
The coverage hole is that Oddbase+even offset is not covered as well as I'd
like.

On Wed, Feb 17, 2021 at 9:16 AM David-Horner <notifications@github.com>
wrote:

...

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

5 participants