Support YJIT for VC++

I heard that supporting YJIT for VC++ needs mmap from k0kubun-san, so I implemented tiny mmap emulation on Windows and committed it to master.

And, I found we need more changes to actually enabled YJIT for VC++, at least:

- YJIT requires OPT_DIRECT_THREADED_CODE or OPT_CALL_THREADED_CODE in rb_yjit_compile_iseq(). Really?
- Maybe ABI differs between VC++ and YJIT's expectation.

Can I get support to fix above?

History

#1 - 12/28/2021 04:31 AM - k0kubun (Takashi Kokubun)
- Assignee set to maximecb (Maxime Chevalier-Boisvert)

#2 - 12/28/2021 05:55 AM - jhawthorn (John Hawthorn)

YJIT requires OPT_DIRECT_THREADED_CODE or OPT_CALL_THREADED_CODE in rb_yjit_compile_iseq().

What option do we use under windows?

Maybe ABI differs between VC++ and YJIT's expectation.

Yes. We would need to switch to the Windows calling convention, which is different enough to be a non-trivial change. It only passes 4 arguments via registers vs 6 in the System V ABI (and also uses different registers) as well as requiring 32 bytes of reserved stack space from the callee ("shadow space").

I think we could likely do this with minimal (but non-trivial) changes by:

- Reserving the 32 bytes of stack "shadow space" in the YJIT entypoint
- Reserving another 16 bytes on the stack in the YJIT entypoint for the 5th and 6th argument
  - This could be an opportunity to do the same and support more arguments on SysV as well
- Conditionally under Windows defining C_ARG_REGS as RCX, RDX, R8, R9, [RSP+32], [RSP+40]
  - Alternatively we could reduce NUM_C_ARG_REGS to 4 and handle the 5th and 6th argument specially where we need them
- Rewriting the two instructions (gen_toregexp/gen_newhash) whose codegen currently modifies the machine stack (we could use a relative stack location instead of push/pop, a callee-saved register, or introduce runtime helper methods)

However I'd like to hear the opinions of YJIT folks when they're back from holidays next week on whether this is the best way to approach this.

#3 - 12/31/2021 06:31 PM - alanwu (Alan Wu)

Yes, supporting Window's x64 calling convention is non-trivial.

In addition to what John already mentioned we also need to uphold unwindability constraints so among other things, longjmp can work. I think longjmp() is used for Ruby exceptions on MSVC like on POSIX.

This means YJIT can't generate PUSH and POP outside of prolog and epilog anymore as they puts the stack pointer temporarily out of alignment. YJIT will also need to supply unwinding info through RtlInstallFunctionTableCallback().

#4 - 01/09/2022 06:09 PM - alanwu (Alan Wu)

- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN)
- Tracker changed from Bug to Feature

#5 - 01/10/2022 11:29 PM - maximecb (Maxime Chevalier-Boisvert)

Supporting Windows is in the plans, but as my colleagues have said it's fairly tricky as it could add a fair bit of complexity in several places. We're
hoping to potentially migrate the YJIT codebase to Rust, which would give us more tools to manage the complexity, and could facilitate a project like this.