Prepending blank module breaks super call in aliased method

Here is the test script:

class A
  def key
    "some_key"
  end
end

module M
  prepend Module.new
  def self.included(base)
    base.alias_method :base_key, :key
  end
  def key
    super + "new_key"
  end
  def generate
    base_key
  end
end

class B < A
  include M
end

x = B.new
p x.generate

In Ruby 2.7 I get the following error:

Traceback (most recent call last):
  2: from test.rb:28:in `<main>'
  1: from test.rb:19:in `generate'
test.rb:15:in `key': super: no superclass method `key' for #<B:0x00007fbc1704d028> (NoMethodError)

If I remove the prepend Module.new line or switch to Ruby 2.6, I get the expected result:

["some_key", "new_key"]

According to bisect, the behavior was changed by 5069c5f5214ce68df8b3954321ad9114c5368dc3.

Could you please check it out?

Assignee set to jeremyevans0 (Jeremy Evans)
This is due to the origin pointers on the module iclasses being incorrectly set to the module's origin instead of the iclass origin. Setting the origin pointers correctly requires using a stack, as the origin iclasses are created after the iclasses themselves. I already did part of the work in the prepend patch in #9573. I've merged the necessary parts of the prepend patch locally and am running tests now. Assuming no errors I'll submit a pull request.

Pull request submitted: https://github.com/ruby/ruby/pull/2978

This includes a GC change to prevent a use-after-free, but I'm not sure if the change introduces a memory leak. Someone that knows more about GC and in what cases iclasses share method tables should probably review.

This pull request also fixes an issue in Module#included_modules to handle origin iclasses for modules correctly.