`emit-stack-sizes`

The tracking issue for this feature is: #54192

The rustc flag `-Z emit-stack-sizes` makes LLVM emit stack size metadata.

> ***NOTE***: *This LLVM feature only supports the ELF object format as of LLVM 8.0. Using this flag with targets that use other object formats (e.g. macOS and Windows) will result in it being ignored.*

Consider this crate:

```
#![crate_type = "lib"]

use std::ptr;

pub fn foo() {
    // this function doesn't use the stack
}

pub fn bar() {
    let xs = [0u32; 2];

    // force LLVM to allocate `xs` on the stack
    unsafe { ptr::read_volatile(&xs.as_ptr()); }
}
```

Using the `-Z emit-stack-sizes` flag produces extra linker sections in the output *object file*.

```
$ rustc -C opt-level=3 --emit=obj foo.rs

$ size -A foo.o
foo.o  :
section                                size   addr
.text                                     0      0
.text._ZN3foo3foo17he211d7b4a3a0c16eE     1      0
.text._ZN3foo3bar17h1acb594305f70c2eE    22      0
.note.GNU-stack                           0      0
.eh_frame                                72      0
Total                                    95

$ rustc -C opt-level=3 --emit=obj -Z emit-stack-sizes foo.rs

$ size -A foo.o
foo.o  :
section                                size   addr
.text                                     0      0
.text._ZN3foo3foo17he211d7b4a3a0c16eE     1      0
.stack_sizes                              9      0
.text._ZN3foo3bar17h1acb594305f70c2eE    22      0
.stack_sizes                              9      0
.note.GNU-stack                           0      0
```

```
.eh_frame                                   72       0
Total                                      113
```

As of LLVM 7.0 the data will be written into a section named `.stack_sizes` and the format is "an array of pairs of function symbol values (pointer size) and stack sizes (unsigned LEB128)".

```
$ objdump -d foo.o

foo.o:     file format elf64-x86-64

Disassembly of section .text._ZN3foo3foo17he211d7b4a3a0c16eE:

0000000000000000 <_ZN3foo3foo17he211d7b4a3a0c16eE>:
   0:   c3                      retq

Disassembly of section .text._ZN3foo3bar17h1acb594305f70c2eE:

0000000000000000 <_ZN3foo3bar17h1acb594305f70c2eE>:
   0:   48 83 ec 10             sub    $0x10,%rsp
   4:   48 8d 44 24 08          lea    0x8(%rsp),%rax
   9:   48 89 04 24             mov    %rax,(%rsp)
   d:   48 8b 04 24             mov    (%rsp),%rax
  11:   48 83 c4 10             add    $0x10,%rsp
  15:   c3                      retq

$ objdump -s -j .stack_sizes foo.o

foo.o:     file format elf64-x86-64

Contents of section .stack_sizes:
 0000 00000000 00000000 00                   .........
Contents of section .stack_sizes:
 0000 00000000 00000000 10                   .........
```

It's important to note that linkers will discard this linker section by default. To preserve the section you can use a linker script like the one shown below.

```
/* file: keep-stack-sizes.x */
SECTIONS
{
  /* `INFO` makes the section not allocatable so it won't be loaded into memory */
  .stack_sizes (INFO) :
  {
    KEEP(*(.stack_sizes));
  }
}
```

The linker script must be passed to the linker using a rustc flag like `-C link-arg`.

```rust
// file: src/main.rs
use std::ptr;

#[inline(never)]
fn main() {
    let xs = [0u32; 2];

    // force LLVM to allocate `xs` on the stack
    unsafe { ptr::read_volatile(&xs.as_ptr()); }
}
```

```console
$ RUSTFLAGS="-Z emit-stack-sizes" cargo build --release

$ size -A target/release/hello | grep stack_sizes || echo section was not found
section was not found

$ RUSTFLAGS="-Z emit-stack-sizes" cargo rustc --release -- \
    -C link-arg=-Wl,-Tkeep-stack-sizes.x \
    -C link-arg=-N

$ size -A target/release/hello | grep stack_sizes
.stack_sizes                              90   176272

$ # non-allocatable section (flags don't contain the "A" (alloc) flag)
$ readelf -S target/release/hello
Section Headers:
  [Nr]   Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
(..)
  [1031] .stack_sizes      PROGBITS         000000000002b090  0002b0f0
       000000000000005a  0000000000000000   L       5     0     1

$ objdump -s -j .stack_sizes target/release/hello

target/release/hello:     file format elf64-x86-64

Contents of section .stack_sizes:
 2b090 c0040000 00000000 08f00400 00000000  ................
 2b0a0 00080005 00000000 00000810 05000000  ................
 2b0b0 00000000 20050000 00000000 10400500  .... ........@..
 2b0c0 00000000 00087005 00000000 00000080  ......p.........
 2b0d0 05000000 00000000 90050000 00000000  ................
 2b0e0 00a00500 00000000 0000               ..........
```

> *Author note: I'm not entirely sure why, in this case, `-N` is required in addition to `-Tkeep-stack-sizes.x`. For example, it's not required when producing statically linked files for the ARM Cortex-M architecture.*