Compromises with large x86-64 binaries

Arthur Eubanks

Link errors with large binaries

```
a.o:(function foo: .text+0x100): relocation
R_X86_64_REX_GOTPCRELX out of range: 2214879970 is not in
[-2147483648, 2147483647]; references 'bar'
```

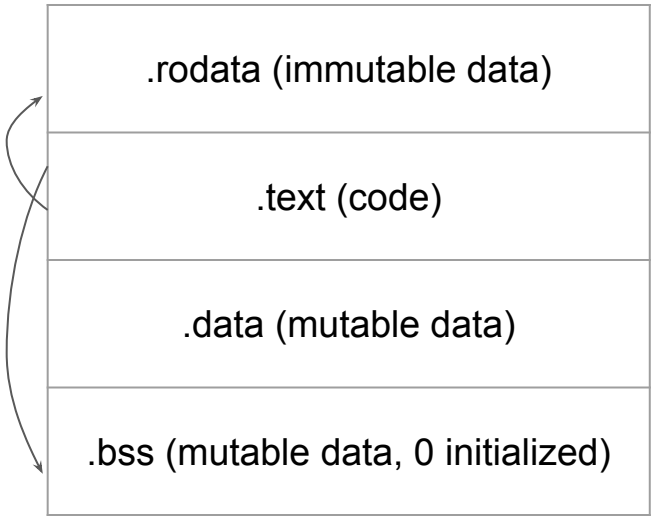
```
relocation R_X86_64_PC32 out of range: 2158227201 is not in
[-2147483648, 2147483647]
```

```
static int i;

int* f() {
    return &i;
}
```

```
Object file:
f:
    lea rax, [rip + i]
    ret

Linked executable:
f:
    lea rax, [rip + 0x2ebd]
    ret
```


.rodata (immutable data)
.text (code)
.data (mutable data)
.bss (mutable data, 0 initialized)

Position Independent Code

`-fpic`

(default nowadays)

```
f:
    lea rax, [rip + 0x2ebd]
    ret
```

`-fno-pic -Wl,--no-pie`

(default back in the day)

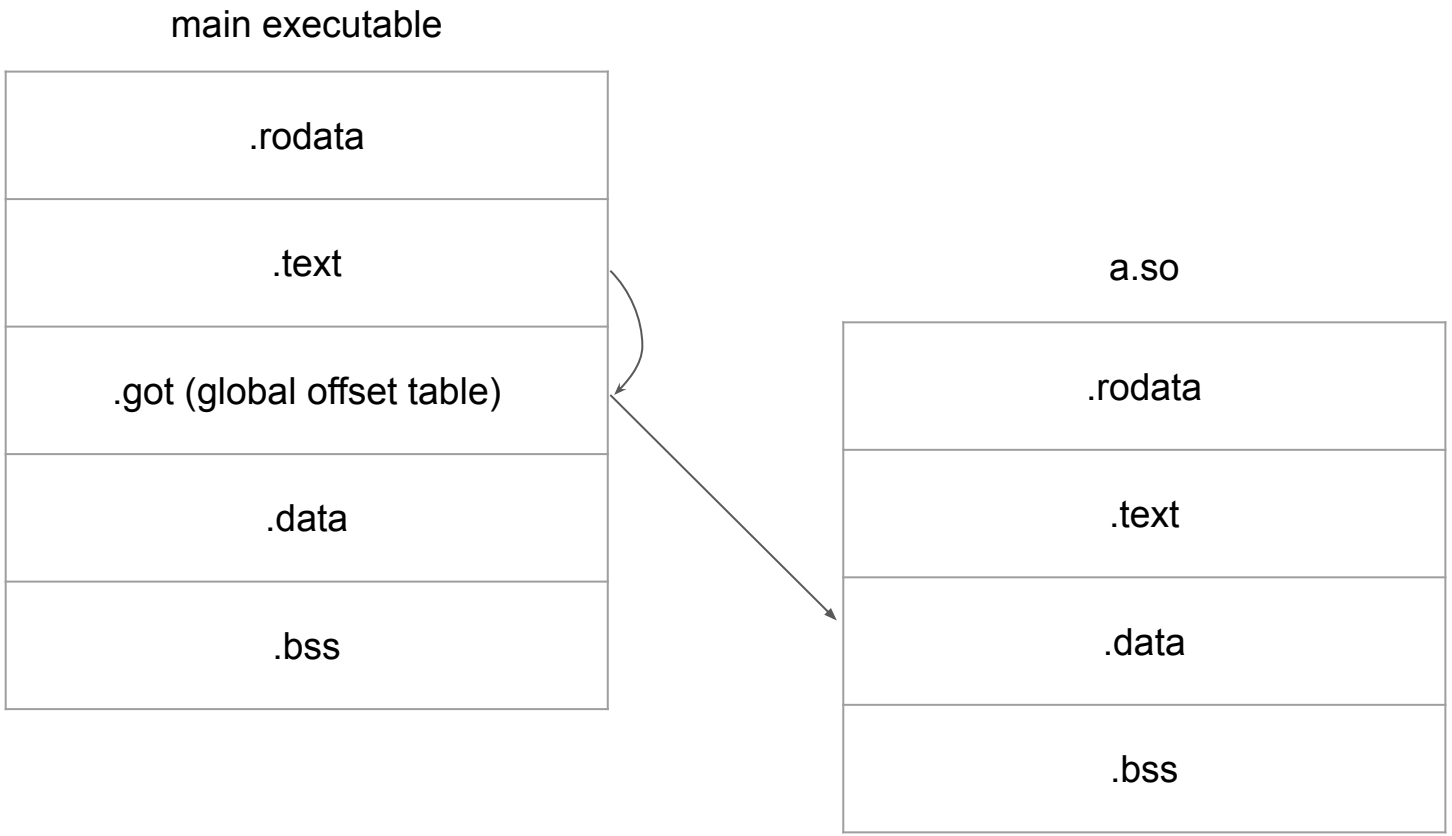
```
f:
    mov eax, 0x404014
    ret
```

```
extern int i;

int* f() {
    return &i;
}
```

```
Object file:
f:
    mov rax, qword ptr [rip + i@GOTPCREL]
    ret

Linked executable:
f:
    mov rax, qword ptr [rip + 0x2e71]
    ret
```


main executable
.rodata
.text
.got (global offset table)
.data
.bss
a.so
.rodata
.text
.data
.bss

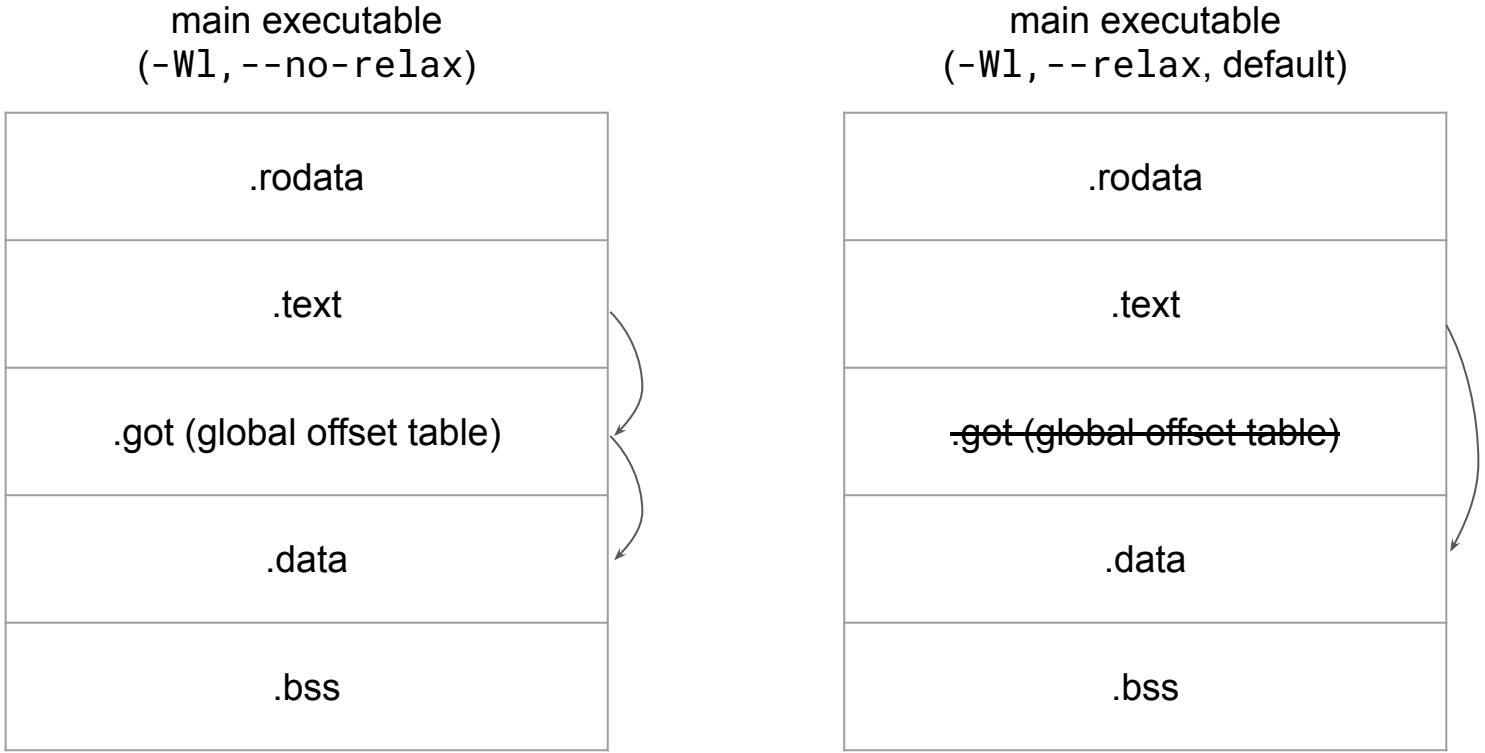
```
extern int i;

int* f() {
    return &i;
}

// b.c
// same executable
int i;
```

```
Object file:
f:
    mov rax, qword ptr [rip + i@GOTPCREL]
    ret

Linked executable:
f:
    mov rax, qword ptr [rip + 0x2e71]
    lea rax, [rip + 0x2ebd]
    ret
```


main executable
(-Wl,--no-relax)
.rodata
.text
.got (global offset table)
.data
.bss
main executable
(-Wl,--relax, default)
.rodata
.text
~~.got (global offset table)~~
.data
.bss

R_X86_64_REX_GOTPCRELX

- lld relaxed unconditionally for simplicity
- Downsides of `-Wl,--no-relax`
  - Extra load when getting address of a global
  - Slightly larger GOT
  - Slightly longer startup time
- D157020 makes this conditional on if rip offset fits in 32-bit signed integer
  - Fixpoint iteration

External globals are fixed!

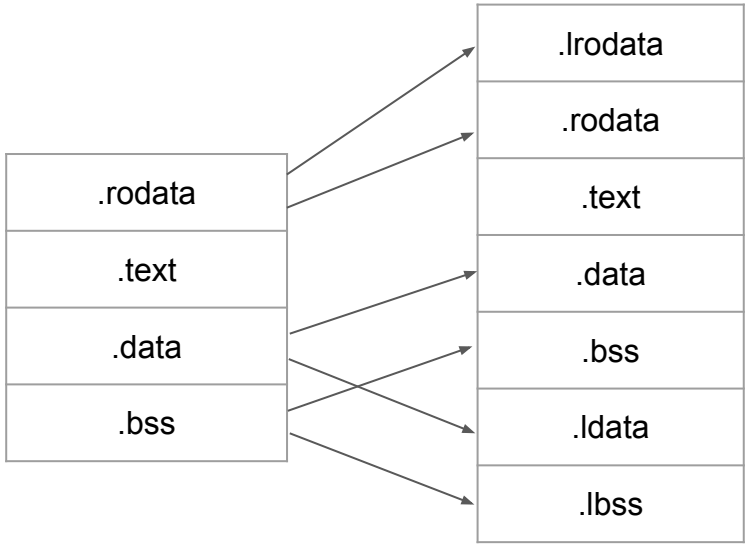
Static globals?

x86-64's medium code model

- Assume data may be further than 2GB from text
  - `lea rax, [rip + i]` won't work
- So instead we add a 64-bit constant from some base
  - ```
    ; get (absolute) address of GOT relative to rip
    lea rcx, [rip + _GLOBAL_OFFSET_TABLE_]
    ; get address of global relative to GOT
    ; offset i@GOTOFF is a link-time constant
    movabs rax, offset i@GOTOFF
    ; get absolute address of global
    add rax, rcx
    ```
- Wow, this is terrible!
    ```

gcc's `-mlarge-data-threshold`

- Split globals into "large" and "small" data based on size of the global
  - Hopefully large data makes up a good portion of binary size
- Place large data farther away from text (`SHF_X86_64_LARGE` section flag)
- Hopefully performance of accessing large data is negligible
- Trade performance for relocation pressure



clang/lld changes

- Place large data sections farther from text (lld)
- Set SHF_X86_64_LARGE flag for data sections in medium/large code model (LLVM codegen)
- Add large data threshold for medium code model (LLVM codegen)
- Add `-mlarge-data-threshold` (Clang)
- Match gcc's default `-mlarge-data-threshold` (Clang)
- Feature parity with gcc's x86-64 (PIC) medium code model!

Future work

- Ability to mark specific globals as large
  - Instrumentation-added globals
- Try making all global references go through GOT
  - Static globals no longer contribute to relocation pressure
- Code size over 2GB?
  - Function calls have the same 32-bit signed integer restriction
  - Large code model exists but is expensive

Thanks!

aeubanks@google.com

https://discourse.llvm.org/