# RISC-V Compact Code Model

Evandro Menezes
evandro.menezes@sifive.com

## INTRODUCTION

Code models are based on the portion of the total address space that may be reached. The native range of instructions usually determines the code models. The range of instructions such as branches and loads may determine different code models to reach code and data in the address space.

When the code or data of a program is within the native range of instructions, it is deemed "small". When the code or data of a program is without the native range of instructions, it is deemed "large". Code models are sometimes defined as:

| Code models | Small code | Large code |
|---|---|---|
| Small data | Small model | Medium model |
| Large data | Compact model | Large model |

RISC-V Data Models

| Data models | Address space | Addressing range | |
|---|---|---|---|
| | | Code | Data |
| ILP 32 | 4GiB | 4GiB | 4GiB |
| LP64 | 16EiB | 2GiB | 4GiB |

For ILP32, the whole address space is within the native range for both code and data. Therefore, the small code model is sufficient.

For LP64, only 2GiB of the address space is within the native range for code and 4GiB for data. The existing small code models, `medlow` & `medany`, are not sufficient.

RISC-V LP64 Code Models

| Code models | 2GiB code | 16EiB code |
|---|---|---|
| 4GiB data | Small model | Medium model |
| 16EiB data | Compact model | Large model |

In order to access more than 2GiB of code and 4GiB of data, the native instructions are not sufficient:

- More than 2GiB of code and 4GiB of data can only be reached indirectly via a 64 bit register.
- 64 bit constants are not natively supported by the RV64I instructions.
- Forming 64 bit constants using several instructions is not optimal.

Thus, supporting the whole address space of 16EiB of code adds prohibitive costs to call sites. Therefore, neither the medium nor the large code models are feasible.

However, the cost to support the address space of 16EiB of data may be acceptable. This compact code model is proposed below.

## RV64 Compact Model

Though this proposal for a compact code model includes the case when there is more than 4GiB of initialized data, it is not the case that is primarily addressed by the new specification. Rather, accessing data anywhere in the whole address space is the primary case addressed by it.

If the global variable has local scope, it is allocated in or adjacent to the global data area. The limit for data of local scope is 2GiB.

If the global variable has global scope, then its literal is allocated in the GOT entry with a corresponding `R_RISCV_64` relocation. When the program is loaded, the dynamic linker should

process this relocation. In the absence of a loader, the linker records the symbol value in the GOT entry.

These new relocation functions are added to the assembler:

- `%gprel_hi(<symbol>)`, `%gprel_lo(<symbol>)`: offset from the global pointer to the symbol.
- `%got_gprel_hi(<symbol>)`, `%got_gprel_lo(<symbol>)`: offset from the global pointer to the GOT entry for the symbol.
- `%gprel(<symbol>)`, `%got_gprel(<symbol>)`: note instruction for relaxation purposes.

These new relocation types are added:

- `R_RISCV_GPREL_HI20`: `S - GP + A`[1]
- `R_RISCV_GPREL_LO20_I`: `S - GP + A`[2]
- `R_RISCV_GPREL_LO20_S`: `S - GP + A`
- `R_RISCV_GPREL_ADD`: (relaxation purposes)
- `R_RISCV_GPREL_LOAD`: (relaxation purposes)
- `R_RISCV_GPREL_STORE`: (relaxation purposes)
- `R_RISCV_GOT_GPREL_HI20`: `G - GP + A`
- `R_RISCV_GOT_GPREL_LO12_I`: `G - GP + A`
- `R_RISCV_GOT_GPREL_ADD`: (relaxation purposes)

---

[1] Legend for the relocation calculations:
- A: the addend used to compute the value of the relocatable field.
- G: the offset into the GOT where the value of the symbol will reside.
- GP: represents the address of the global data area.
- P: the place (section offset or address) of the relocation.
- S: the value of the symbol.

[2] The new relocation types `R_RISCV_GPREL_LO20_I` and `R_RISCV_GPREL_LO20_S` are used instead of the existing `R_RISCV_GPREL_I` and `R_RISCV_GPREL_S` because of potentially different use by the linker currently.

- `R_RISCV_GOT_GPREL_LOAD`: (relaxation purposes)
- `R_RISCV_GOT_GPREL_STORE`: (relaxation purposes)
- `R_RISCV_64_PCREL`: `S - P + A`

## Compact Model Coding Examples

### Function Prolog

Functions in shared objects that access the global data area must setup the `gp` register using this code sequence, for example:

```
auipc gp, %pcrel_hi(__global_pointer__)
addi  gp, gp, %pcrel_lo(__global_pointer__)
ld    t0, 0(gp)
add   gp, gp, t0
…
.section .text.__global_pointer__, "aMG", @progbits, 8, __global_pointer__, comdat
.align 3
.hidden __global_pointer__
.type __global_pointer__, object
__global_pointer__:
.quad __global_pointer$ - .
```

The expression "`__global_pointer$ - .`" results in the relocation type `R_RISCV_64_PCREL`. This relocation is position independent, provided that the information available to the linker matches the run time environment, therefore it can be used in both executable and shared objects.

Note that the code example above places the literal for the global data area in a `comdat` section that is shared by all references to this literal in the object.

### Data Objects

The code examples below assume that the `gp` register points to the global data area.

Note that the GOT is used for both non PIC and PIC objects.

| Source | Assembly | Relocations |
|---|---|---|
| extern int src;<br>extern int dst;<br>extern void *ptr;<br>static int lsrc;<br>static int ldst;<br>static void foo(void); | .extern src<br>.extern dst<br>.extern ptr<br>.comm .Llsrc, 4<br>.comm .Lldst, 4<br>.local foo | R_RISCV_64<br>R_RISCV_64<br>R_RISCV_64 |
| | .text<br>... // setup gp | |
| dst = src; | lui t0, %got_gprel_hi(src)<br>add t0, t0, gp, %got_gprel(src)<br>ld t0, %got_gprel_lo(src)(t0)<br>lw t2, 0(t0), %got_gprel(src)<br>lui t1, %got_gprel_hi(dst)<br>add t1, t1, gp, %got_gprel(dst)<br>ld t1, %got_gprel_lo(dst)(t1)<br>sw t2, 0(t1), %got_gprel(dst) | R_RISCV_GOT_GPREL_HI20<br>R_RISCV_GOT_GPREL_ADD<br>R_RISCV_GOT_GPREL_LO12_I<br>R_RISCV_GOT_GPREL_LOAD<br>R_RISCV_GOT_GPREL_HI20<br>R_RISCV_GOT_GPREL_ADD<br>R_RISCV_GOT_GPREL_LO12_I<br>R_RISCV_GOT_GPREL_STORE |
| ptr = &src; | lui t0, %got_gprel_hi(src)<br>add t0, t0, gp, %got_gprel(src)<br>ld t0, %got_gprel_lo(src)(t0)<br>lui t1, %got_gprel_hi(ptr)<br>add t1, t1, gp, %got_gprel(ptr)<br>ld t1, %got_gprel_lo(ptr)(t1)<br>sd t0, 0(t1), %got_gprel(ptr) | R_RISCV_GOT_GPREL_HI20<br>R_RISCV_GOT_GPREL_ADD<br>R_RISCV_GOT_GPREL_LO12_I<br>R_RISCV_GOT_GPREL_HI20<br>R_RISCV_GOT_GPREL_ADD<br>R_RISCV_GOT_GPREL_LO12_I<br>R_RISCV_GOT_GPREL_STORE |
| ldst = &lsrc; | 1:<br>lui t0, %gprel_hi(.Llsrc)<br>add t0, t0, gp, %gprel(1b)<br>addi t0, t0, %gprel_lo(.Llsrc)<br>2:<br>lui t1, %gprel_hi(.Lldst)<br>add t1, t1, gp, %gprel(2b)<br>sd t0, %gprel_lo(.Lldst)(t1) | <br>R_RISCV_GPREL_HI20<br>R_RISCV_GPREL_ADD<br>R_RISCV_GPREL_LO20_I<br><br>R_RISCV_GPREL_HI20<br>R_RISCV_GPREL_ADD<br>R_RISCV_GPREL_LO20_S |
| ptr = foo; | la t0, foo<br><br>lui t1, %got_gprel_hi(ptr)<br>add t1, t1, gp, %got_gprel(ptr)<br>ld t1, %got_gprel_lo(ptr)(t1)<br>sd t0, 0(t1), %got_gprel(ptr) | R_RISCV_PCREL_HI20<br>R_RISCV_PCREL_LO12_I<br>R_RISCV_GOT_GPREL_HI20<br>R_RISCV_GOT_GPREL_ADD<br>R_RISCV_GOT_GPREL_LO12_I<br>R_RISCV_GOT_GPREL_STORE |

### RELAXATION

The cost of the compact code model can be quite significant, so it is important to minimize this cost when the conditions allow for it. This can be done at link time through the addition of special relocation types to allow it. Additionally, additional assembly macros simplify adding such special relocations.

This macro is used to produce the literal for a local symbol:

```
lla   <rd>, %gprel(<symbol>)
```

Which expands to:

```
lui   <rd>, %gprel_hi(<symbol>)                // R_RISCV_GPREL_HI20 (symbol)
add   <rd>, <rd>, gp, %gprel(<symbol>)         // R_RISCV_GPREL_ADD (symbol)
addi  <rd>, <rd>, %gprel_lo(<symbol>)          // R_RISCV_GPREL_LO20_I (symbol)
```

If the literal for the symbol is allocated within the global data area, then this sequence may be relaxed to:

```
addi  <rd>, gp, %gprel_lo(<symbol>)            // R_RISCV_GPREL_LO20_I (symbol)
```

For a global symbol, this macro is used to produce its literal:

```
la    <rd>, %got_gprel(<symbol>)
```

Which expands to:

```
lui   <rd>, %got_gprel_hi(<symbol>)            // R_RISCV_GOT_GPREL_HI20 (symbol)
add   <rd>, <rd>, gp, %got_gprel(<symbol>)     // R_RISCV_GOT_GPREL_ADD (symbol)
ld    <rd>, %got_gprel_lo(<symbol>)(<rd>)      // R_RISCV_GOT_GPREL_LO20_I (symbol)
```

If the GOT entry for the literal is allocated in the vicinity of the global data area, then this sequence may be relaxed to:

```
ld    <rd>, %got_gprel_lo(<symbol>)(gp)        // R_RISCV_GOT_GPREL_LO20_I (symbol)
```

If the global symbol is allocated in the executable, then this macro is equivalent to:

```
lla   <rd>, %gprel(<symbol>)
```

For loading from or storing to a local symbol, these macros are used:

```
l{b|h|w|d}  <rd>, %gprel(<symbol>)
s{b|h|w|d}  <rd>, %gprel(<symbol>), <rt>
```

Which expand to:

```
lui   <rd>, %gprel_hi(<symbol>)                // R_RISCV_GPREL_HI20 (symbol)
add   <rd>, <rd>, gp, %gprel(<symbol>)         // R_RISCV_GPREL_ADD (symbol)
l{b|h|w|d} <rd>, %gprel_lo(<symbol>)(<rd>)     // R_RISCV_GPREL_LO20_I (symbol)
```

Or:

```
lui   <rt>, %gprel_hi(<symbol>)                // R_RISCV_GPREL_HI20 (symbol)
add   <rt>, <rt>, gp, %gprel(<symbol>)         // R_RISCV_GPREL_ADD (symbol)
s{b|h|w|d} <rd>, %gprel_lo(<symbol>)(<rt>)     // R_RISCV_GPREL_LO20_S (symbol)
```

If the symbol is allocated within the global data area, then these sequences may be relaxed to:

```
l{b|h|w|d} <rd>, %gprel_lo(<symbol>)(gp)       // R_RISCV_GPREL_LO20_I (symbol)
```

Or:

```
s{b|h|w|d} <rd>, %gprel_lo(<symbol>)(gp)       // R_RISCV_GPREL_LO20_S (symbol)
```

For loading from or storing to a global symbol, these macros are used:

```
l{b|h|w|d}  <rd>, <offset>(<rt>), %got_gprel(<symbol>)
s{b|h|w|d}  <rd>, <offset>(<rt>), %got_gprel(<symbol>)
```

Which expand to:

```
l{b|h|w|d}  <rd>, <offset>(<rt>)               // R_RISCV_GOT_GPREL_LOAD (symbol)
s{b|h|w|d}  <rd>, <offset>(<rt>)               // R_RISCV_GOT_GPREL_STORE (symbol)
```

If the global symbol is allocated and referenced within the global data area of the executable, then these macros are equivalent to:

```
l{b|h|w|d} <rd>, %gprel_lo(<symbol> + <offset>)(gp) // R_RISCV_GPREL_LO20_I (symbol)
```

Or:

```
s{b|h|w|d} <rd>, %gprel_lo(<symbol> + <offset>)(gp) // R_RISCV_GPREL_LO20_S (symbol)
```

The table below demonstrates the results of relaxation when the global is allocated and referenced in the executable:

| Source | Assembly | Relaxed |
|---|---|---|
| `extern int src;`<br>`extern int dst;`<br>`extern void *ptr;`<br>`static void foo(void);` | `.extern src`<br>`.extern dst`<br>`.extern ptr`<br>`.local foo` | |
| | `.text`<br>`...  // setup gp` | |

| Source | Assembly | Relaxed |
|---|---|---|
| `dst = src;` | `lui t0, %got_gprel_hi(src)`<br>`add t0, t0, gp, %got_gprel(src)`<br>`ld t0, %got_gprel_lo(src)(t0)`<br>`lw t2, 0(t0), %got_gprel(src)`<br>`lui t1, %got_gprel_hi(dst)`<br>`add t1, t1, gp, %got_gprel(dst)`<br>`ld t1, %got_gprel_lo(dst)(t1)`<br>`sw t2, 0(t1), %got_gprel(dst)` | `lui t0, %gprel_hi(src)`<br>`add t0, t0, gp`<br>`addi t0, t0, %gprel_lo(src)`<br>`lw t2, 0(t0)`<br>`lui t1, %gprel_hi(dst)`<br>`add t1, t1, gp`<br>`addi t1, t1, %gprel_lo(dst)`<br>`sw t2, 0(t1)` |
| `ptr = &src;` | `lui t0, %got_gprel_hi(src)`<br>`add t0, t0, gp, %got_gprel(src)`<br>`ld t0, %got_gprel_lo(src)(t0)`<br>`lui t1, %got_gprel_hi(ptr)`<br>`add t1, t1, gp, %got_gprel(ptr)`<br>`ld t1, %got_gprel_lo(ptr)(t1)`<br>`sd t0, 0(t1), %got_gprel(ptr)` | `lui t0, %gprel_hi(src)`<br>`add t0, t0, gp`<br>`addi t0, t0, %gprel_lo(src)`<br>`lui t1, %gprel_hi(ptr)`<br>`add t1, t1, gp`<br>`addi t1, t1, %gprel_lo(ptr)`<br>`sd t0, 0(t1)` |
| `ptr = foo;` | `la t0, foo`<br>`lui t1, %got_gprel_hi(ptr)`<br>`add t1, t1, gp, %got_gprel(ptr)`<br>`ld t1, %got_gprel_lo(ptr)(t1)`<br>`sd t0, 0(t1), %got_gprel(ptr)` | `la t0, foo`<br>`lui t1, %gprel_hi(ptr)`<br>`add t1, t1, gp`<br>`addi t1, t1, %gprel_lo(ptr)`<br>`sd t0, 0(t1)` |

The table below demonstrates the results of relaxation when the GOT entry for the global is in the vicinity of the global data area:

| Source | Assembly | Relaxed |
|---|---|---|
| `extern int src;`<br>`extern int dst;`<br>`extern void *ptr;`<br>`static void foo(void);` | `.extern src`<br>`.extern dst`<br>`.extern ptr`<br>`.local foo` | |
| | `.text`<br>`... // setup gp` | |
| `dst = src;` | `lui t0, %got_gprel_hi(src)`<br>`add t0, t0, gp, %got_gprel(src)`<br>`ld t0, %got_gprel_lo(src)(t0)`<br>`lw t2, 0(t0), %got_gprel(src)`<br>`lui t1, %got_gprel_hi(dst)`<br>`add t1, t1, gp, %got_gprel(dst)`<br>`ld t1, %got_gprel_lo(dst)(t1)`<br>`sw t2, 0(t1), %got_gprel(dst)` | `ld t0, %got_gprel_lo(src)(gp)`<br>`lw t2, 0(t0)`<br>`ld t1, %got_gprel_lo(dst)(gp)`<br>`sw t2, 0(t1)` |
| `ptr = &src;` | `lui t0, %got_gprel_hi(src)`<br>`add t0, t0, gp, %got_gprel(src)`<br>`ld t0, %got_gprel_lo(src)(t0)`<br>`lui t1, %got_gprel_hi(ptr)`<br>`add t1, t1, gp, %got_gprel(ptr)`<br>`ld t1, %got_gprel_lo(ptr)(t1)`<br>`sd t0, 0(t1), %got_gprel(ptr)` | `ld t0, %got_gprel_lo(src)(gp)`<br>`ld t1, %got_gprel_lo(ptr)(gp)`<br>`sd t0, 0(t1)` |

| Source | Assembly | Relaxed |
|---|---|---|
| ptr = foo; | la t0, foo<br>lui t1, %got_gprel_hi(ptr)<br>add t1, t1, gp, %got_gprel(ptr)<br>ld t1, %got_gprel_lo(ptr)(t1)<br>sd t0, 0(t1), %got_gprel(ptr) | la t0, foo<br><br><br>ld t1, %got_gprel_lo(ptr)(gp)<br>sd t0, 0(t1) |

The table below demonstrates the results of relaxation when the global is allocated and referenced in the global data area of the executable:

| Source | Assembly | Relaxed |
|---|---|---|
| extern int src;<br>extern int dst;<br>extern void *ptr;<br>static int lsrc;<br>static int ldst;<br>static void foo(void); | .extern src<br>.extern dst<br>.extern ptr<br>.comm .Llsrc, 4<br>.comm .Lldst, 4<br>.local foo | |
| | .text<br>... // setup gp | |
| dst = src; | lui t0, %got_gprel_hi(src)<br>add t0, t0, gp, %got_gprel(src)<br>ld t0, %got_gprel_lo(src)(t0)<br>lw t2, 0(t0), %got_gprel(src)<br>lui t1, %got_gprel_hi(dst)<br>add t1, t1, gp, %got_gprel(dst)<br>ld t1, %got_gprel_lo(dst)(t1)<br>sw t2, 0(t1), %got_gprel(dst) | <br><br>addi t0, gp, %gprel_lo(src)<br>lw t2, %gprel_lo(src)(gp)<br><br><br>addi t1, gp, %gprel_lo(dst)<br>sw t2, %gprel_lo(dst)(gp) |
| ptr = &src; | lui t0, %got_gprel_hi(src)<br>add t0, t0, gp, %got_gprel(src)<br>ld t0, %got_gprel_lo(src)(t0)<br>lui t1, %got_gprel_hi(ptr)<br>add t1, t1, gp, %got_gprel(ptr)<br>ld t1, %got_gprel_lo(ptr)(t1)<br>sd t0, 0(t1), %got_gprel(ptr) | <br><br>addi t0, gp, %gprel_lo(src)<br><br><br>addi t1, gp, %gprel_lo(ptr)<br>sd t0, %gprel_lo(ptr)(gp) |
| ldst = &lsrc; | lui t0, %gprel_hi(.Llsrc)<br>add t0, t0, gp, %gprel(.Llsrc)<br>addi t0, t0, %gprel_lo(.Llsrc)<br>lui t1, %gprel_hi(.Lldst)<br>add t1, t1, gp, %gprel(.Lldst)<br>sd t0, %gprel_lo(.Lldst)(t1) | <br><br>addi t0, gp, %gprel_lo(.Llsrc)<br><br>addi t1, gp, %gprel_lo(.Lldst)<br>sd t0, %gprel_lo(.Lldst)(gp) |
| ptr = foo; | la t0, foo<br>lui t1, %got_gprel_hi(ptr)<br>add t1, t1, gp, %got_gprel(ptr)<br>ld t1, %got_gprel_lo(ptr)(t1)<br>sd t0, 0(t1), %got_gprel(ptr) | la t0, foo<br><br><br>addi t1, gp, %gprel_lo(ptr)<br>sd t0, %gprel_lo(ptr)(gp) |