Introduction to RISC-V ABI

PLCT Tech Report

Sinan Lin 林思南
sinan.lin@aalto.fi

Contents of table

Step 1. Write down source code

main.c

```c
// main.c
#include "add.h"

void _start() {
    asm("li a0, 155\n"
        "li a1, 100\n"
        "call add\n"
        "li a7, 93\n"
        "ecall\n"
    );
}
```

add.h

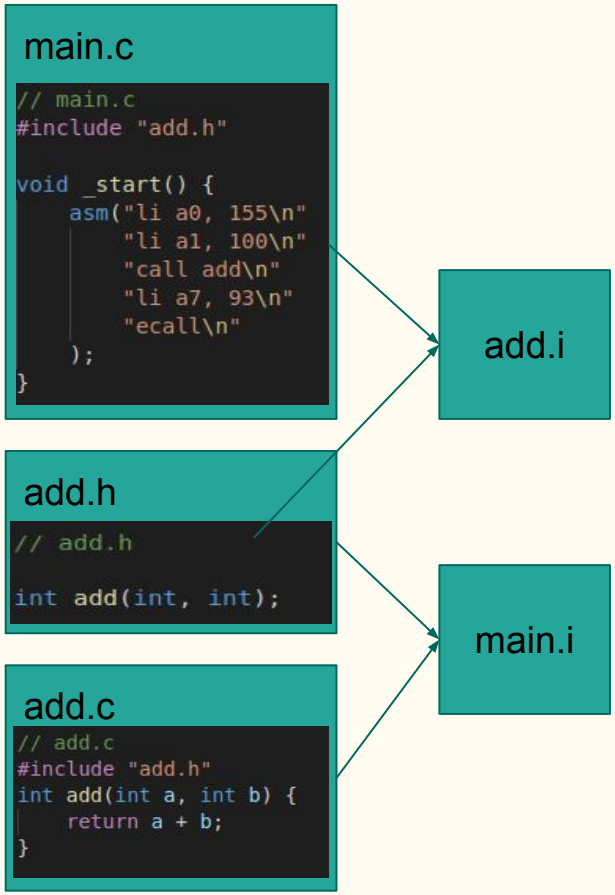
```c
// add.h

int add(int, int);
```

add.c

```c
// add.c
#include "add.h"
int add(int a, int b) {
    return a + b;
}
```

sys_exit(add(100, 155))

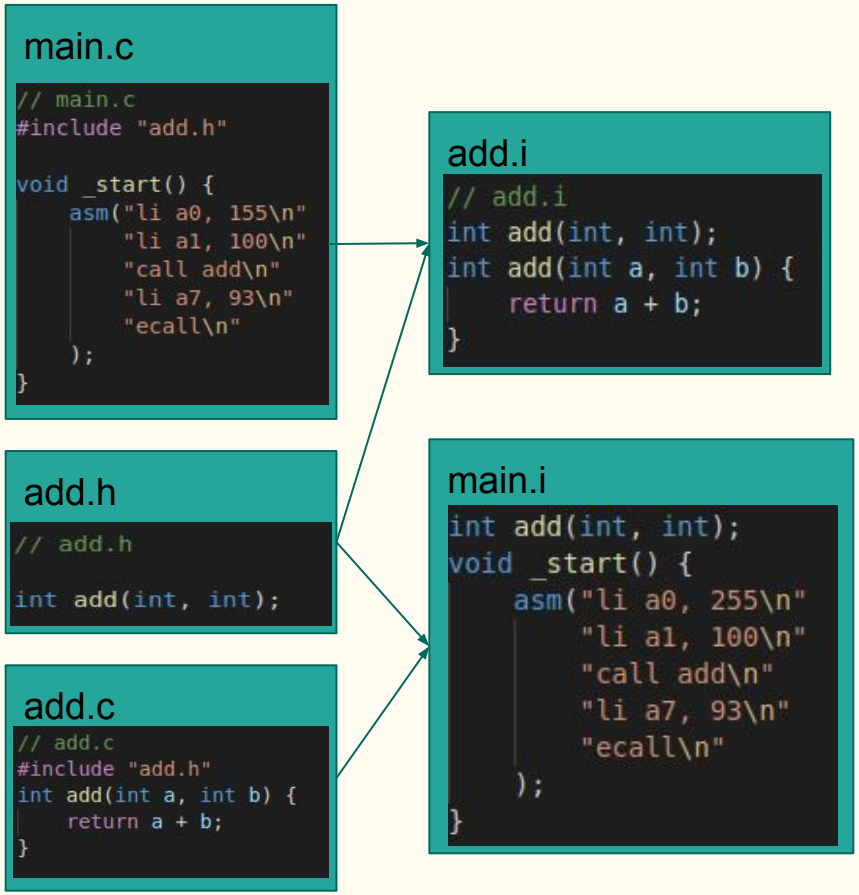
Step 2. Use preprocessor to expand macro



Preprocessor expands macros

```
riscv64-unknown-linux-gnu-cpp -P main.c -o main.i
riscv64-unknown-linux-gnu-cpp -P add.c  -o add.i
```

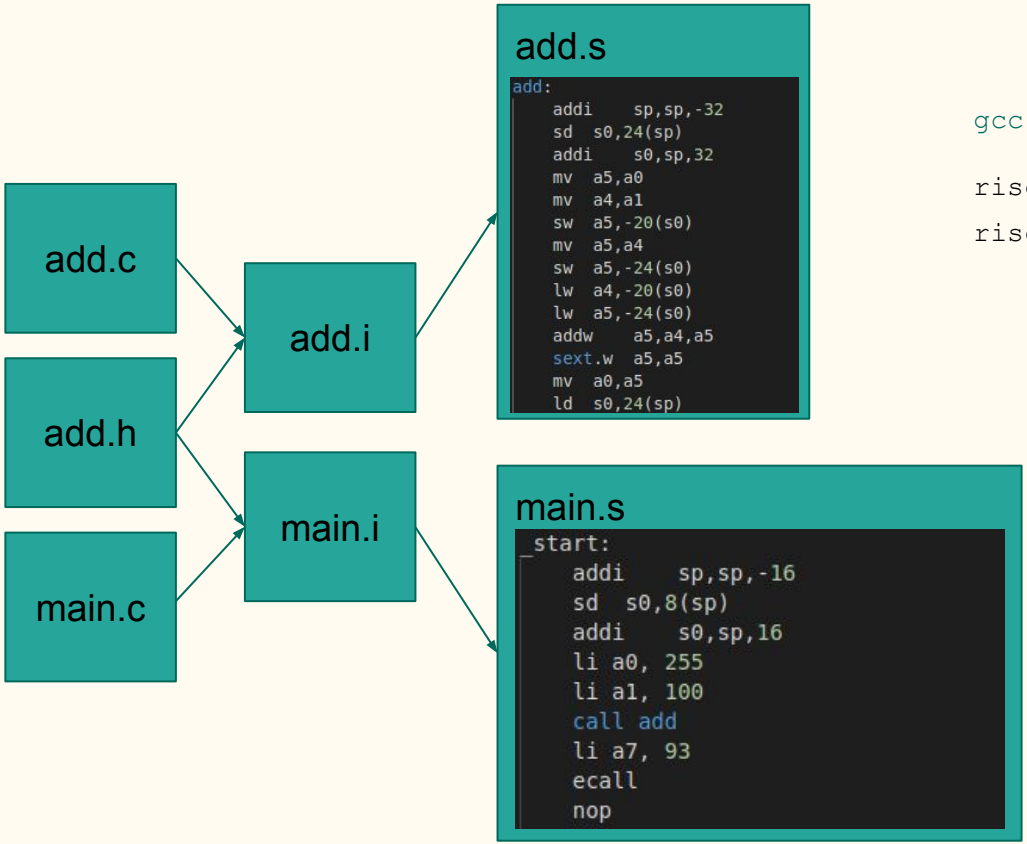
Step 2. Use preprocessor to expand macro



Preprocessor expands macros

```
riscv64-unknown-linux-gnu-cpp -P main.c -o main.i
riscv64-unknown-linux-gnu-cpp -P add.c  -o add.i
```

Step 3. GCC compiles source code to assembly



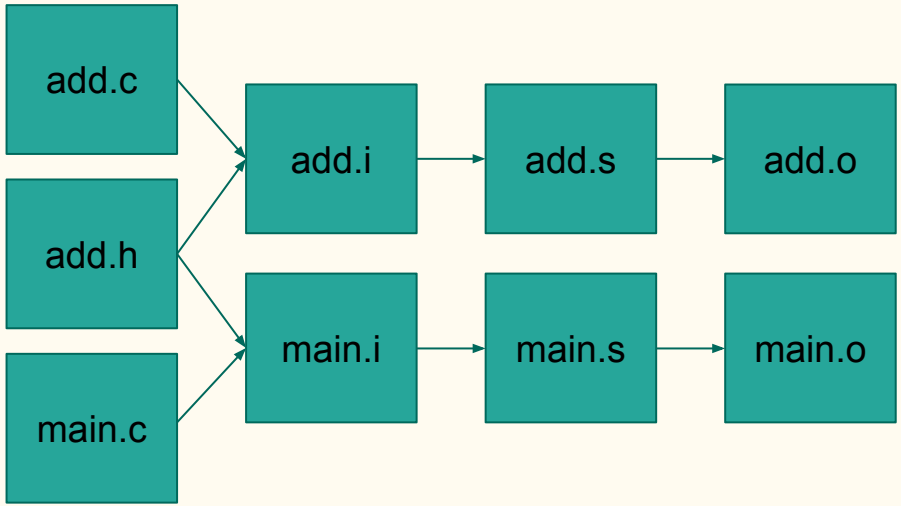
gcc generates assembly

```
riscv64-unknown-linux-gnu-gcc -S main.i -o main.s
riscv64-unknown-linux-gnu-gcc -S add.i -o add.s
```

Step 4. Assembler converts code to object file

as converts assembly code into object code

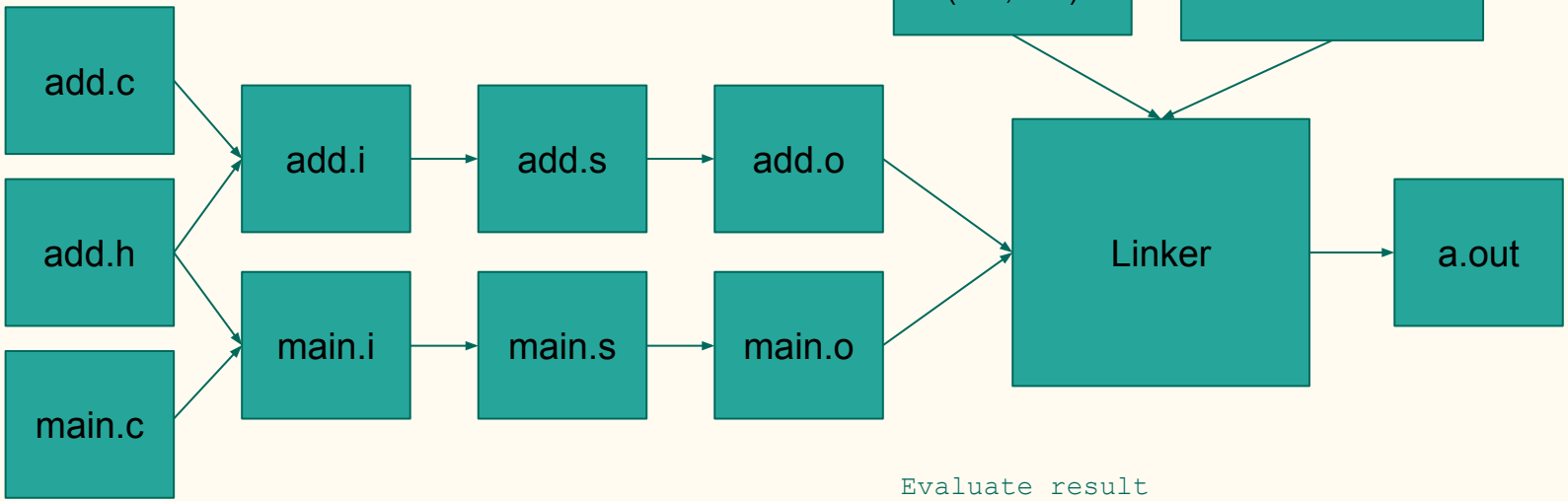
```
riscv64-unknown-linux-gnu-as main.s -o main.o
riscv64-unknown-linux-gnu-as add.s  -o add.o
```




```
add.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <add>:
   0:   fe010113                addi    sp,sp,-32
main.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <_start>:
   0:   ff010113                addi    sp,sp,-16
   4:   00813423                sd      s0,8(sp)
   8:   01010413                addi    s0,sp,16
   c:   0ff00513                li      a0,255
  10:   06400593                li      a1,100
  14:   00000097                auipc   ra,0x0
  18:   000080e7                jalr    ra # 14 <_start+0x14>
  1c:   05d00893                li      a7,93
  20:   00000073                ecall
  24:   00000013                nop
  28:   00813403                ld      s0,8(sp)
  2c:   01010113                addi    sp,sp,16
  30:   00008067                ret
```

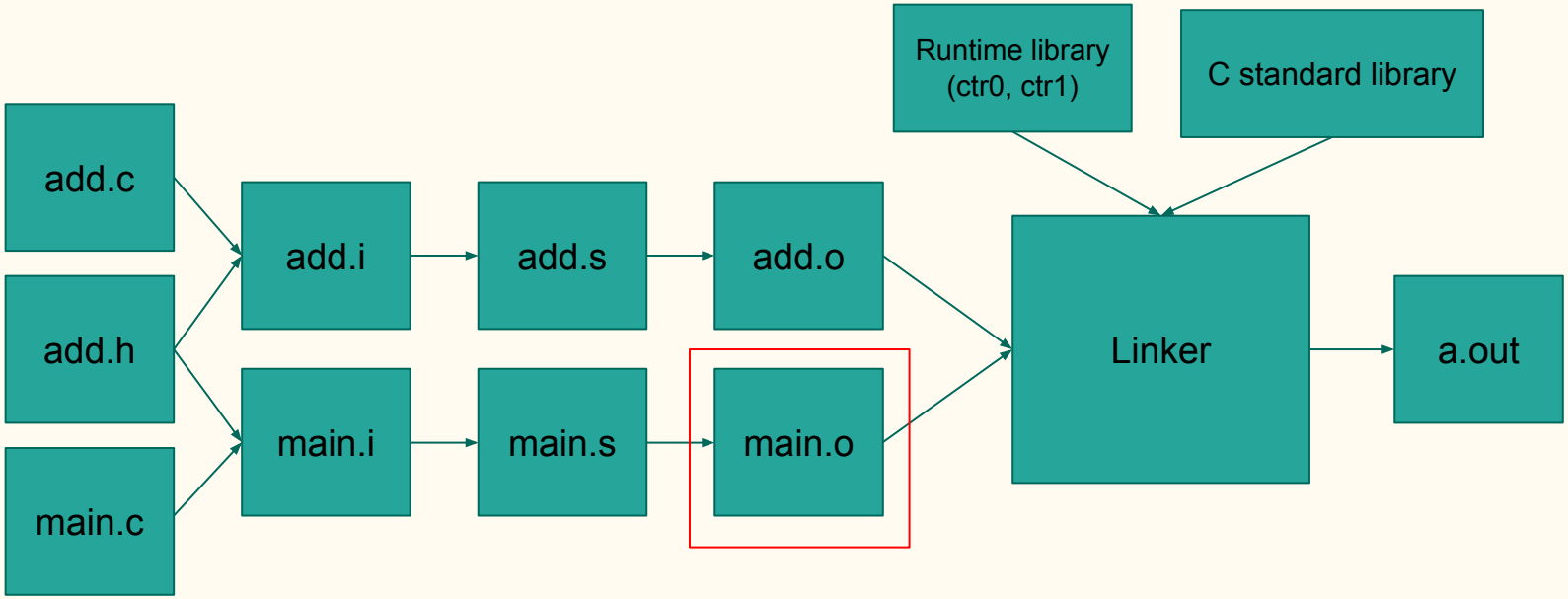
Step 5. Linker combines object files and emits executable file

```
ld converts object files into executable file

riscv64-unknown-linux-gnu-ld -static main.o add.o
```



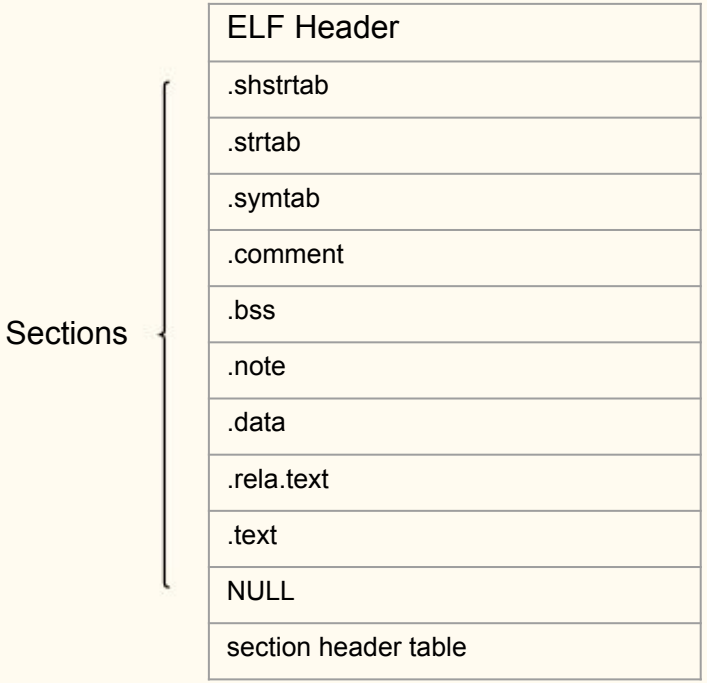

```
Evaluate result
qemu-riscv64 a.out
echo $?
>> 255
```

- RISC-V specification in ELF format
- RISC-V ABI



ELF (Executable and linkable file format)

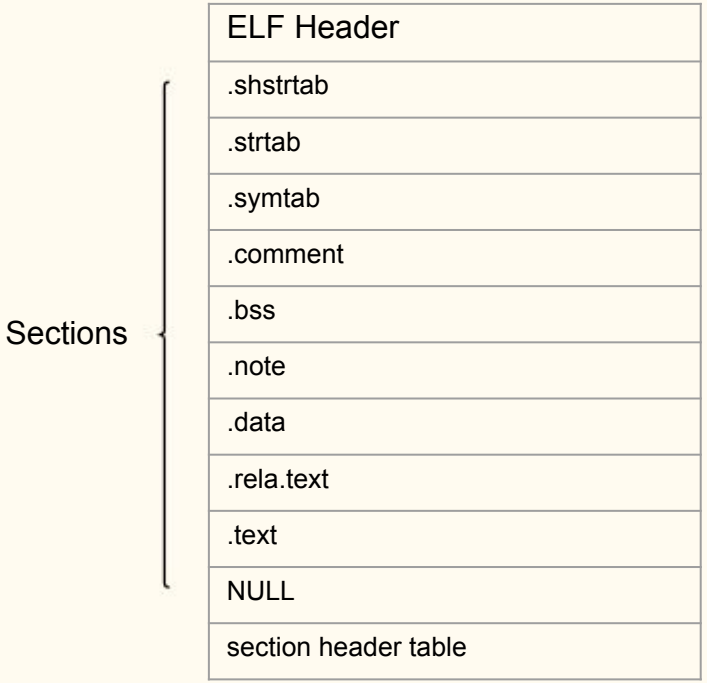
- ELF Header
  - General information about the binary
- Sections
  - Information needed for linking object files in order to build an executable
- Segments
  - Chunks of information to prepare the executable to be loaded into memory.



Layout of main.o

ELF (Executable and linkable file format)

- ELF Header
  - General information about the binary
- Sections
  - Information needed for linking object files in order to build an executable
- Segments
  - Chunks of information to prepare the executable to be loaded into memory.



Layout of main.o

ELF Header

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          496 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         10
  Section header string table index: 9
```

main.o

```
typedef struct
{
  unsigned char e_ident[16];    /* Magic number and other info */
  uint16_t  e_type;         /* Object file type */
  uint16_t  e_machine;      /* Architecture */
  uint32_t  e_version;      /* Object file version */
  uint64_t  e_entry;        /* Entry point virtual address */
  uint64_t  e_phoff;        /* Program header table file offset */
  uint64_t  e_shoff;        /* Section header table file offset */
  uint32_t  e_flags;        /* Processor-specific flags */
  uint16_t  e_ehsize;       /* ELF header size in bytes */
  uint16_t  e_phentsize;    /* Program header table entry size */
  uint16_t  e_phnum;        /* Program header table entry count */
  uint16_t  e_shentsize;    /* Section header table entry size */
  uint16_t  e_shnum;        /* Section header table entry count */
  uint16_t  e_shstrndx;     /* Section header string table index */
} Elf64_Ehdr;
```

riscv-glibc/elf/elf.h

ELF Header with RISC-V specification

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          496 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         10
  Section header string table index: 9
```

`readelf -h main.o`

RVC allows instructions to be aligned to 16-bit boundaries, the linker is permitted to use RVC instructions such as C.JAL in the relaxation process. (‘C’ Standard Extension)

ILP64D: Integer calling-convention with hardware floating-point calling convention for FLEN=64

e_machine

- EM_RISCV (243) for RISC-V ELF

e_flags

Bit 0	Bit 1 - 2	Bit 3	Bit 4	Bit 5 - 31
RVC	Float ABI	RVE	TSO	*Reserved*

- EF_RISCV_RVC (0x0001)
- EF_RISCV_FLOAT_ABI_SOFT (0x0000)
- EF_RISCV_FLOAT_ABI_SINGLE (0x0002)
- EF_RISCV_FLOAT_ABI_DOUBLE (0x0004)
- EF_RISCV_FLOAT_ABI_QUAD (0x0006)
- EF_RISCV_FLOAT_ABI (0x0006)
- EF_RISCV_RVE (0x0008)
- EF_RISCV_TSO (0x0010)

Section header table

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          496 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         10
  Section header string table index: 9
```

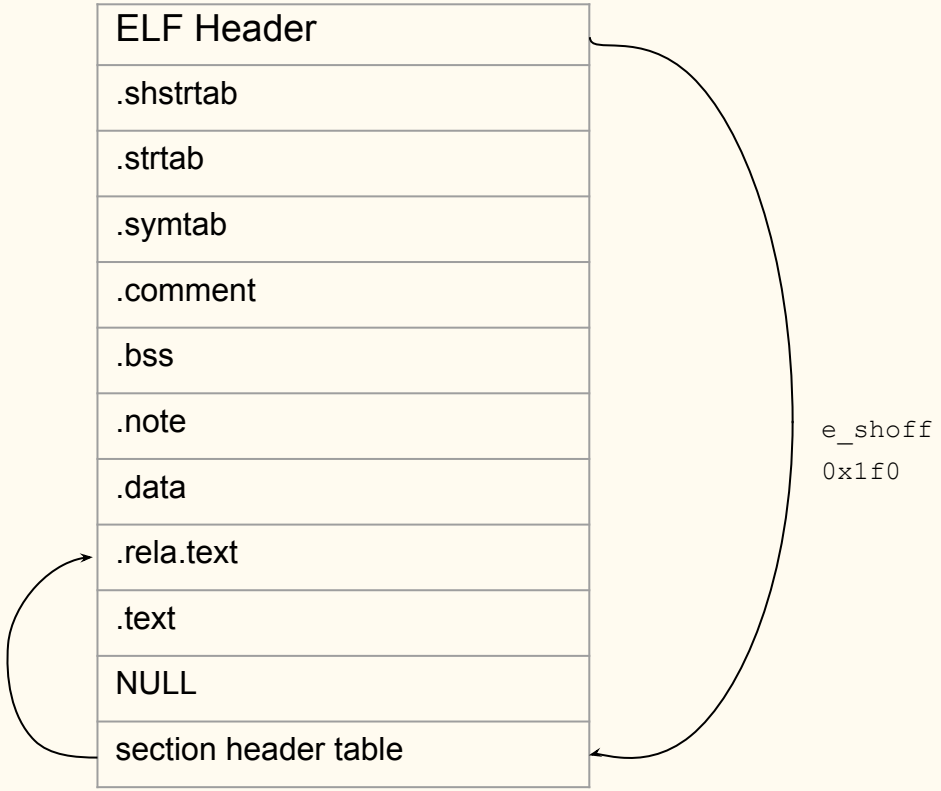
0x1F0

Sections

ELF Header
.shstrtab
.strtab
.symtab
.comment
.bss
.note
.data
.rela.text
.text
NULL
section header table

Layout of main.o

Table look-up for .rela.text section



Layout of main.o

```
e_shoff - (2 * e_shentsize)
= 0x170
```

Sections

- Name
- Type
- Virtual address
- File Offset
- Size
- Flag
- Alignment

```
typedef struct
{
  uint32_t  sh_name;        /* Section name (string tbl index) */
  uint32_t  sh_type;        /* Section type */
  uint64_t  sh_flags;       /* Section flags */
  uint64_t  sh_addr;        /* Section virtual addr at execution */
  uint64_t  sh_offset;      /* Section file offset */
  uint64_t  sh_size;        /* Section size in bytes */
  uint32_t  sh_link;        /* Link to another section */
  uint32_t  sh_info;        /* Additional section information */
  uint64_t  sh_addralign;   /* Section alignment */
  uint64_t  sh_entsize;     /* Entry size if section holds table */
} Elf64_Shdr;
```

riscv-glibc/elf/elf.h

Relocation section

Name Type Virtual address Link Info

```
Section Headers:
  [Nr] Name              Type            Address          Offset     Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000 00000000 0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS        0000000000000000 00000040 0000000000000026  0000000000000000  AX       0     0     2
  [ 2] .rela.text        RELA            0000000000000000 00000170 0000000000000030  0000000000000018   I      7     1     8
  [ 3] .data             PROGBITS        0000000000000000 00000066 0000000000000000  0000000000000000  WA       0     0     1
  [ 4] .bss              NOBITS          0000000000000000 00000066 0000000000000000  0000000000000000  WA       0     0     1
  [ 5] .comment          PROGBITS        0000000000000000 00000066 0000000000000013  0000000000000001  MS       0     0     1
  [ 6] .note.GNU-stack   PROGBITS        0000000000000000 00000079 0000000000000000  0000000000000000           0     0     1
  [ 7] .symtab           SYMTAB          0000000000000000 00000080 00000000000000d8  0000000000000018           8     7     8
  [ 8] .strtab           STRTAB          0000000000000000 00000158 0000000000000013  0000000000000000           0     0     1
  [ 9] .shstrtab         STRTAB          0000000000000000 000001a0 000000000000004a  0000000000000000           0     0     1
```

`readelf -S main.o`

Relocation table entry

- location where a relocation will take place
- index of symbol
- relocation type
- addend

```
Relocation section '.rela.text' at offset 0x170 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000000e  000800000012 R_RISCV_CALL      0000000000000000 add + 0
00000000000e  000000000033 R_RISCV_RELAX                        0
```

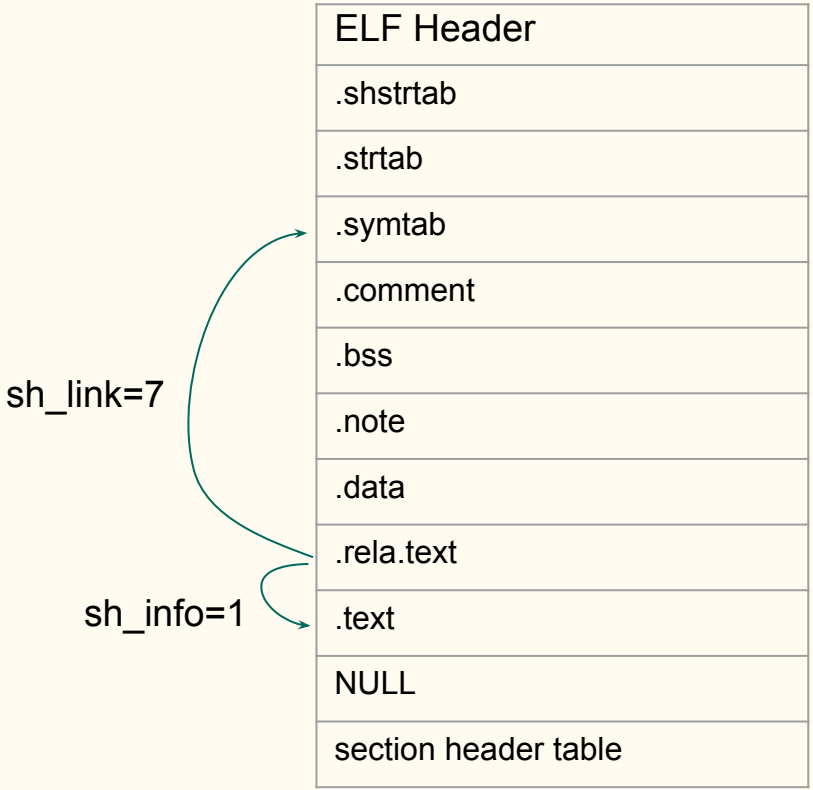
rela.text section in main.o

```
/* Relocation table entry with addend (in section of type SHT_RELA).  */
typedef struct
{
  uint64_t  r_offset;       /* Address */
  uint64_t  r_info;         /* Relocation type and symbol index */
  int64_t   r_addend;       /* Addend */
} Elf64_Rela;
```

`riscv-glibc/elf/elf.h`

Table look-up in .rela.text section

the relocation section is linked to two other sections. One of them is the symbol table, where the symbols that will be relocated are held.



Relocation table entry

- location where a relocation will take place
- index of symbol
- relocation type
- addend

```
Relocation section '.rela.text' at offset 0x170 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000000e  000800000012 R_RISCV_CALL      0000000000000000 add + 0
00000000000e  000000000033 R_RISCV_RELAX                        0
```

rela.text section in main.o

```
/* Relocation table entry with addend (in section of type SHT_RELA).  */
typedef struct
{
  uint64_t  r_offset;       /* Address */
  uint64_t  r_info;         /* Relocation type and symbol index */
  int64_t   r_addend;       /* Addend */
} Elf64_Rela;
```

`riscv-glibc/elf/elf.h`

Relocation type

Enum	ELF Reloc Type	Description	Field	Calculation	Details
0	R_RISCV_NONE	None			
1	R_RISCV_32	Runtime relocation	*word32*	S + A	
2	R_RISCV_64	Runtime relocation	*word64*	S + A	
3	R_RISCV_RELATIVE	Runtime relocation	*wordclass*	B + A	
4	R_RISCV_COPY	Runtime relocation			Must be in executable; not allowed in shared library
5	R_RISCV_JUMP_SLOT	Runtime relocation	*wordclass*	S	Handled by PLT unless `LD_BIND_NOW`
6	R_RISCV_TLS_DTPMOD32	TLS relocation	*word32*	S->TLSINDEX	

Relocations type in RISC-V

- 0~58 Defined
- 59~255 Reserved

Relocation type

Enum	ELF Reloc Type	Description	Field	Calculation	Details
18	R_RISCV_CALL	PC-relative call	*J-Type*	S + A - P	Macros `call`, `tail`
51	R_RISCV_RELAX	Previous reloc can be relaxed			

Relocations type *R_RISCV_CALL and R_RISCV_RELEX*

J type field:

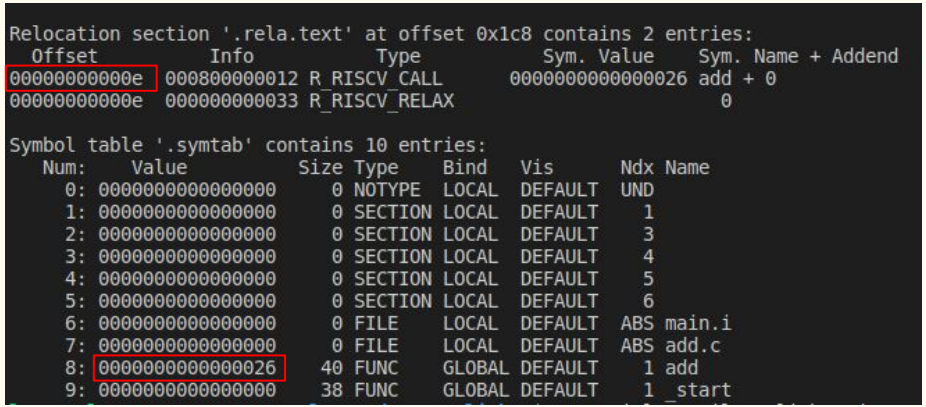
- the immediate field in a J-type instruction

R_RISCV_CALL is associated with pairs of instructions (AUIPC+JALR) generated by the CALL or TAIL pseudoinstructions.

```
auipc x1,      <hi20bits>
jalr   x0, x1, <lo12bits>
```

Calculation symbols

```
Concat main.o and add.o
riscv64-unknown-linux-gnu-ld -relocatable main.o add.o -o main_concat.o
riscv64-unknown-linux-gnu-readelf -rs main_concat.o
```



```
Relocation section '.rela.text' at offset 0x1c8 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000000e  000800000012 R_RISCV_CALL      0000000000000026 add + 0
00000000000e  000000000033 R_RISCV_RELAX                        0

Symbol table '.symtab' contains 10 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     6: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.i
     7: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS add.c
     8: 0000000000000026    40 FUNC    GLOBAL DEFAULT    1 add
     9: 0000000000000000    38 FUNC    GLOBAL DEFAULT    1 _start
```

Calculation Symbols

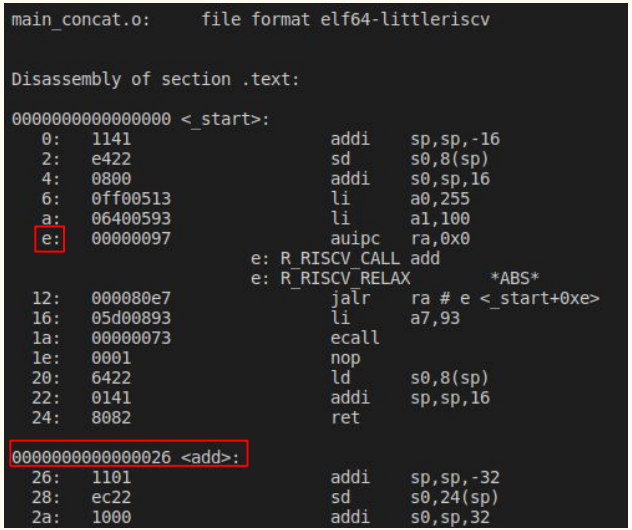
The following table provides details on the variables used in relocation calculation:

Variable	Description
A	Addend field in the relocation entry associated with the symbol
P	Position of the relocation
S	Value of the symbol in the symbol table

P = 0xe
S = 0x26
A = 0x0

S+A-P = 0x18

Symbols calculation for R_RISCV_CALL



P = 0xe
S = 0x26
A = 0x0

S+A-P = 0x18

```
riscv64-unknown-linux-gnu-objdump -Dr main_concat.o
```

R_RISCV_RELEX

Linker optimization -- Relocation Relaxation

```
main_concat.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <_start>:
   0:   1141                    addi    sp,sp,-16
   2:   e422                    sd      s0,8(sp)
   4:   0800                    addi    s0,sp,16
   6:   0ff00513                li      a0,255
   a:   06400593                li      a1,100
   e:   00000097                auipc   ra,0x0
                        e: R_RISCV_CALL add
                        e: R_RISCV_RELAX        *ABS*
  12:   000080e7                jalr    ra # e <_start+0xe>
  16:   05d00893                li      a7,93
  1a:   00000073                ecall
  1e:   0001                    nop
  20:   6422                    ld      s0,8(sp)
  22:   0141                    addi    sp,sp,16
  24:   8082                    ret

0000000000000026 <add>:
  26:   1101                    addi    sp,sp,-32
  28:   ec22                    sd      s0,24(sp)
  2a:   1000                    addi    s0,sp,32
```

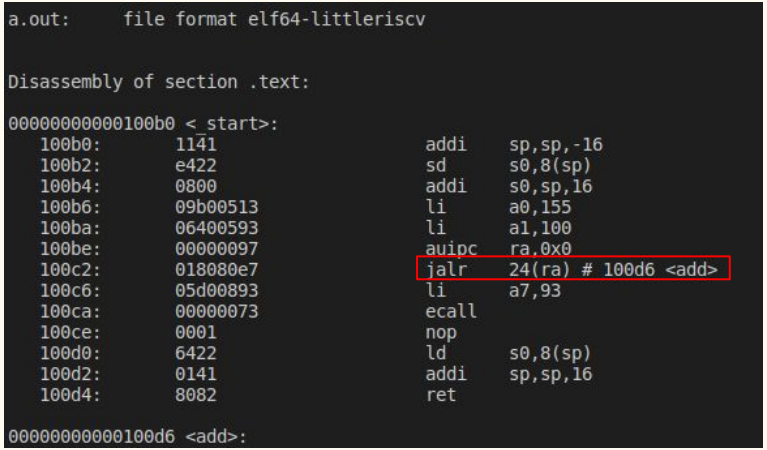
riscv64-unknown-linux-gnu-objdump -Dr main_concat.o

```
a.out:     file format elf64-littleriscv


Disassembly of section .text:

00000000000100b0 <_start>:
   100b0:       1141                    addi    sp,sp,-16
   100b2:       e422                    sd      s0,8(sp)
   100b4:       0800                    addi    s0,sp,16
   100b6:       09b00513                li      a0,155
   100ba:       06400593                li      a1,100
   100be:       014000ef                jal     ra,100d2 <add>
   100c2:       05d00893                li      a7,93
   100c6:       00000073                ecall
   100ca:       0001                    nop
   100cc:       6422                    ld      s0,8(sp)
   100ce:       0141                    addi    sp,sp,16
   100d0:       8082                    ret

00000000000100d2 <add>:
```

riscv64-unknown-linux-gnu-objdump -Dr a.out

Without R_RISCV_RELEX

```
Compile without linker optimization with flag -mno-relax
riscv64-unknown-linux-gnu-gcc -c -mno-relax main.c -o main.o
riscv64-unknown-linux-gnu-gcc -c add.c -o add.o
riscv64-unknown-linux-gnu-ld main.o add.o
```



0x18

riscv64-unknown-linux-gnu-objdump -Dr a.out

What’s the address of entry point?

```
lnan95@lnan95-P65-67HSHP:~/leetcode/test_linker$ riscv64-unknown-linux-gnu-ld -verbose | grep _start
ENTRY(_start)
  PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x10000)); . = SEGMENT_START("text-segment", 0x10000) + SIZEOF_HEADERS;
      PROVIDE_HIDDEN (__rela_iplt_start = .);
     PROVIDE_HIDDEN (__tdata_start = .);
    PROVIDE_HIDDEN (__preinit_array_start = .);
    PROVIDE_HIDDEN (__init_array_start = .);
    PROVIDE_HIDDEN (__fini_array_start = .);
    __bss_start = .;
```

Program entry address

ABI (Application Binary Interface)

- High-level ABI (Itanium C++ ABI)
  - Name demangling
  - Data Layout
- Generic ABI (UNIX System V ABI)
  - System call
  - Function Calling Sequence
- Processor-specific ABI (RISC-V psABI)
  - Register Convention
  - Procedure Calling Convention
  - Code models
  - C type details

```
_start:
    addi    sp,sp,-16
    sd  s0,8(sp)
    addi    s0,sp,16
    li a0, 255
    li a1, 100
    call add
    li a7, 93
    ecall
    nop
```

main.s

Name demangling in C++ ABI

call `_Z3addii`

`Riscv64-unknown-linux-gnu-c++filt _Z3addii`
>> add(int, int)

Register Convention

Integer Register Convention

Name	ABI Mnemonic	Meaning	Preserved across calls?
x0	zero	Zero	-- (Immutable)
x1	ra	Return address	No
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	-- (Unallocatable)
x4	tp	Thread pointer	-- (Unallocatable)
x5-x7	t0-t2	Temporary registers	No
x8-x9	s0-s1	Callee-saved registers	Yes
x10-x17	a0-a7	Argument registers	No
x18-x27	s2-s11	Callee-saved registers	Yes
x28-x31	t3-t6	Temporary registers	No

Floating-point Register Convention

Name	ABI Mnemonic	Meaning	Preserved across calls?
f0-f7	ft0-ft7	Temporary registers	No
f8-f9	fs0-fs1	Callee-saved registers	Yes*
f10-f17	fa0-fa7	Argument registers	No
f18-f27	fs2-fs11	Callee-saved registers	Yes*
f28-f31	ft8-ft11	Temporary registers	No

RISC-V psABI: Integer Register Convention

frame pointer: s0

integer registers tp and gp should not be modified

Named ABIs

Default ABI

This specification defines the following named ABIs:

- **ILP32**: Integer calling-convention only, hardware floating-point calling convention is not used (i.e. ELFCLASS32 and EF_RISCV_FLOAT_ABI_SOFT).
- **ILP32F**: ILP32 with hardware floating-point calling convention for FLEN=32 (i.e. ELFCLASS32 and EF_RISCV_FLOAT_ABI_SINGLE).
- **ILP32D**: ILP32 with hardware floating-point calling convention for FLEN=64 (i.e. ELFCLASS32 and EF_RISCV_FLOAT_ABI_DOUBLE).
- **ILP32E**: ILP32E calling-convention only, hardware floating-point calling convention is not used (i.e. ELFCLASS32, EF_RISCV_FLOAT_ABI_SOFT, and EF_RISCV_RVE).
- **LP64**: Integer calling-convention only, hardware floating-point calling convention is not used (i.e. ELFCLASS64 and EF_RISCV_FLOAT_ABI_SOFT).
- **LP64F**: LP64 with hardware floating-point calling convention for FLEN=32 (i.e. ELFCLASS64 and EF_RISCV_FLOAT_ABI_SINGLE).
- **LP64D**: LP64 with hardware floating-point calling convention for FLEN=64 (i.e. ELFCLASS64 and EF_RISCV_FLOAT_ABI_DOUBLE).
- **LP64Q**: LP64 with hardware floating-point calling convention for FLEN=128 (i.e. ELFCLASS64 and EF_RISCV_FLOAT_ABI_QUAD).

RISC-V ELF psABI specification named ABI

C type details

- **LP64, LP64F, LP64D, and LP64Q**: use the following type sizes and alignments (based on the LP64 convention):

Type	Size (Bytes)	Alignment (Bytes)
bool/_Bool	1	1
char	1	1
short	2	2
int	4	4
long	8	8
long long	8	8
__int128	16	16
void *	8	8
float	4	4
double	8	8
long double	16	16
float _Complex	8	4
double _Complex	16	8
long double _Complex	32	16

C type sizes and alignments in RISC-V 64

Procedure Calling Convention

- Integer Calling Convention
- Hardware Floating-point Calling Convention

Integer Calling Convention

The base integer calling convention provides eight argument registers, a0-a7, the first two of which are also used to return values.

size < XLEN passed in a single argument register

Aggregate /Scalars

```
li   a0, imm
call func
```

XLEN=32bits for ILP32* ABI
XLEN=64bits for ILP64* ABI

*Assume registers are available

Integer Calling Convention

The base integer calling convention provides eight argument registers, a0-a7, the first two of which are also used to return values.

Aggregate /Scalars	size <= XLEN	passed in a single argument register
	size in (XLEN, 2*XLEN]	passed in a pair of argument registers for low bits and high bits

```
ld  a0, %lo
ld  a1, %hi
call  func
```

XLEN=32bits for ILP32* ABI
XLEN=64bits for ILP64* ABI

*Assume registers are available

Integer Calling Convention

The base integer calling convention provides eight argument registers, a0-a7, the first two of which are also used to return values.

Aggregate /Scalars	size <= XLEN	passed in a single argument register
	size in (XLEN, 2*XLEN]	passed in a pair of argument registers for low bits and high bits
	size > 2*XLEN	passed by reference and are replaced in the argument list with the address

XLEN=32bits for ILP32* ABI
XLEN=64bits for ILP64* ABI

*Assume registers are available

Hardware Floating-point Calling Convention

The hardware floating-point calling convention adds eight floating-point argument registers, fa0-fa7, the first two of which are also used to return values.

Scalars	size <= FLEN	passed in a single argument floating point register
	Otherwise	passed according to the integer calling convention

FLEN is the width of a floating-point register in the ABI
ILP64 and ILP32 ABI do not support hardware floating-point calling convention

*Assume registers are available

Hardware Floating-point Calling Convention

The hardware floating-point calling convention adds eight floating-point argument registers, fa0-fa7, the first two of which are also used to return values.

Aggregate with mixed variables	One floating point	passed as though it were a standalone floating-point real
	Two floating point	passed in two floating-point registers
	One floating-point and one integer	passed in a floating-point register and an integer register
	Otherwise	passed according to the integer calling convention

*Assume registers are available, and sizes of fp are all less or equal then FLEN

Code models

The code model determines what these addressing templates look like, and thus which relocations are emitted.

- Small
- Medium(default)
- Compact

Recall:

```
main_concat.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <_start>:
   0:   1141                    addi    sp,sp,-16
   2:   e422                    sd      s0,8(sp)
   4:   0800                    addi    s0,sp,16
   6:   0ff00513                li      a0,255
   a:   06400593                li      a1,100
   e:   00000097                auipc   ra,0x0
                        e: R_RISCV_CALL add
                        e: R_RISCV_RELAX        *ABS*
  12:   000080e7                jalr    ra # e <_start+0xe>
  16:   05d00893                li      a7,93
  1a:   00000073                ecall
  1e:   0001                    nop
  20:   6422                    ld      s0,8(sp)
  22:   0141                    addi    sp,sp,16
  24:   8082                    ret

0000000000000026 <add>:
  26:   1101                    addi    sp,sp,-32
  28:   ec22                    sd      s0,24(sp)
  2a:   1000                    addi    s0,sp,32
```

`riscv64-unknown-linux-gnu-objdump -Dr main_concat.o`

Code models

- Small
  - address the whole RV32 address space or the lower 2 GiB of the RV64 address space
  - `lui` and `ld` or `st` instruction pair
- Medium
- Compact

```
long global_symbol[2];

int main() {
  return global_symbol[0] != 0;
}

// -mcmodel=medlow
lui      a5,%hi(global_symbol)
ld       a0,%lo(global_symbol)(a5)
snez  a0,a0
```

Relocation Type:
R_RISCV_HI20
R_RISCV_LO12_I

```
Relocation section '.rela.text' at offset 0x178 contains 4 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000006  00070000001a R_RISCV_HI20      0000000000000000 global_symbol + 0
000000000006  000000000033 R_RISCV_RELAX                        0
00000000000a  00070000001b R_RISCV_LO12_I    0000000000000000 global_symbol + 0
00000000000a  000000000033 R_RISCV_RELAX                        0
```

Code models

- Small
  - address the whole RV32 address space or the lower 2 GiB of the RV64 address space
  - `lui` and `ld` or `st` instruction pair
- Medium
  - address the range between -2 GiB and +2 GiB from its position
  - `auipc` and `ld or st`
- Compact

```
long global_symbol[2];

int main() {
   return global_symbol[0] != 0;
}

// -mcmodel=medany
.LA0: auipc  a5, %pcrel_hi(global_symbol)
addi    a5,a5,%pcrel_lo(.LA0)
ld      a5,0(a5)
snez  a5,a5
```

Relocation Type:
R_RISCV_PCREL_HI20
R_RISCV_PCREL_LO12_I

```
Relocation section '.rela.text' at offset 0x198 contains 4 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000006  000800000017 R_RISCV_PCREL_HI2 0000000000000000 global_symbol + 0
000000000006  000000000033 R_RISCV_RELAX                        0
00000000000a  000600000018 R_RISCV_PCREL_LO1 0000000000000006 .L0  + 0
00000000000a  000000000033 R_RISCV_RELAX                        0
```

Code models

- Small
  - address the whole RV32 address space or the lower 2 GiB of the RV64 address space
  - `lui` and `ld` or `st` instruction pair
- Medium
  - address the range between -2 GiB and +2 GiB from its position
  - `auipc` and `ld or st`
- Compact
  - address the whole 64-bit address space
  - `lui` and `addi`

Reference

1. The RISC-V Instruction Set Manual Volume I: User-Level ISA, page 67
2. RISC-V ELF psABI specification
3. SYSTEM V APPLICATION BINARY INTERFACE MIPS RISC Processor, page 4-16
4. CppCon 2019: Louis Dionne “The C++ ABI From the Ground Up”
5. CppCon 2017: Michael Spencer “My Little Object File: How Linkers Implement C++”
6. Itanium C++ ABI
7. All Aboard, Part 4: The RISC-V Code Models