Run ILP32 on RV64

廖仕华 PLCT LAB
shihua@iscas.ac.cn

2023/09/23

报告内容

- 什么是ABI
- ILP32 On 64bit
- Run ILP32 on RV64

什么是ABI

Q1：为什么C++ Primer上，在讲述int类型的时候，指出int至少要有16位位宽，且不能超过64位位宽？

Q2：为什么在Windows下，我用MinGW编译的二进制文件不能和Visual Studio 2015编译的链接在一起？

Q3：为什么MSVC编译的程序不能同时在x86架构下的MacOS和Windows 10上运行呢？

什么是ABI

ABI（Application Binary Interface）

ABI是操作系统、编程语言、编译器所遵守的一组规则，以期让编译成功后的二进制程序能正确加载、链接、运行。

在我看来，ABI由三个部分组成：

1. 指令集和硬件环境
2. 编程语言、编译器和库
3. 操作系统

什么是ABI

1.指令集和硬件环境

在这里，我们以相对简单的Intel i386为例，链接

什么是ABI

Table 2.1: Scalar Types

Type	C	sizeof	Alignment (bytes)	Intel386 Architecture
Integral	`_Bool`†	1	1	boolean
	`char` `signed char`	1	1	signed byte
	`unsigned char`	1	1	unsigned byte
	`short` `signed short`	2	2	signed twobyte
	`unsigned short`	2	2	unsigned twobyte

	`int`
	`signed int`
	`enum`†††
		4	4	signed fourbyte

	`unsigned int`	4	4	unsigned fourbyte
	`long` `signed long`	4	4	signed fourbyte
	`unsigned long`	4	4	unsigned fourbyte
	`long long` `signed long long`	8	4	signed eightbyte
	`unsigned long long`	8	4	unsigned eightbyte
Pointer	*any-type* `*` *any-type* `(*)()`	4	4	unsigned fourbyte
Floating-point	`float`	4	4	single (IEEE-754)
	`double` `long double`††††	8	4	double (IEEE-754)
	`__float80`†† `long double`††††	12	4	80-bit extended (IEEE-754)
	`__float128`††	16	16	128-bit extended (IEEE-754)

什么是ABI

1.指令集和硬件环境

在这里，我们以相对简单的Intel i386为例，链接

什么是ABI

Table 2.2: Stack Frame with Base Pointer

Position	Contents	Frame
`4n+8(%ebp)`	memory argument fourbyte n	Previous
	...	
`8(%ebp)`	memory argument fourbyte 0	
`4(%ebp)`	return address	Current
`0(%ebp)`	previous `%ebp` value	
`-4(%ebp)`	unspecified	
	...	
`0(%esp)`	variable size	

什么是ABI

Table 2.3: Register Usage

Register	Usage	Preserved across function calls
%eax	scratch register; also used to return integer and pointer values from functions; also stores the address of a returned struct or union	No
%ebx	callee-saved register; also used to hold the GOT pointer when making function calls via the PLT	Yes
%ecx	scratch register	No
%edx	scratch register; also used to return the upper 32bits of some 64bit return types	No
%esp	stack pointer	Yes
%ebp	callee-saved register; optionally used as frame pointer	Yes
%esi	callee-saved register	yes
%edi	callee-saved register	yes

什么是ABI

什么是ABI

2.编程语言和编译器、库

以C++为例，目前的C++ ABI分为两大派别：

Intel推动的Itanium C++ ABI

Mircosoft的MSVC ABI

某种意义上来说，C++ ABI又可以戏称为编译器ABI。一般来说，C++ ABI需要编译器决定的部分包括name mangling、异常处理、调用构造/析构函数、Class和虚表的布局与对齐等。

什么是ABI

Itanium C++ ABI

Contents

什么是ABI

3.操作系统

Linux Standard Base

Specifications Archive

The LSB 5 Specification Series

The Linux Standard Base is now on LSB 5. A further evolution of the specification, with considerable internal restructuring, finally delivering on a modular LSB.

The LSB 5.0 Specification

This is the approved final version of the LSB 5.0 specification. The tables below present two popular formats, you may also go to the page with all formats.

Release notes outlining changes from the previous release of the specification can be found at http://www.linuxfoundation.org/en/ReleaseNotes50

Note that the LSB 5.0 Core specification set is an evolution of the ISO/IEC International Standard 23360, which corresponded to LSB 3.1. This edition is not to be considered an ISO standard.

LSB 5.0 was released June 3, 2015.

Notes:

Note: if you are planning to certify conformance to LSB 5.0, please begin by reading the applicable Product Standard.

Green table cells indicate available specs, red unavailable.

Document Set	Functional Area	Architecture															
		Generic		IA32		IA64		PPC32		PPC64		S390		S390X		AMD64	
LSB	Common	HTML	PDF														
	Core	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF
	Desktop	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF	HTML	PDF
	Runtime Languages	HTML	PDF														
	Imaging	HTML	PDF														
Trial Use	Gtk3, Graphics	HTML	PDF														

ILP32 On 64bit

ILP32 -> int = long = pointer = 32

LP64 -> long = pointer = 64

	ILP32	LP32	LLP64	LP64	ILP64
char	8	8	8	8	8
short	16	16	16	16	16
int	32	32	32	32	64
long	32	32	32	64	64
long long	64	64	64	64	64
void*	32	32	64	64	64

ILP32 On 64bit

“当我编译一个使用不到 4 GB RAM 的程序时，拥有 64 位指针绝对是愚蠢的。当这样的指针值出现在结构中时，它们不仅浪费了一半的内存，而且有效地丢弃了一半的缓存。”　　—— Knuth

目前，出于不同的原因，x86_64推出过x32 ABI、mips推出过n32 ABI，aarch64 推出过 ilp32 ABI。这三个不同架构的ABI都试图在64位机器上运行32位的指针。

ILP32 On 64bit

x86_64 推出x32 ABI的主要目的是为了提高在x86_64下的程序运行效率和内存开销。

aarch64 推出ilp32 ABI的主要目的是为了解决aarch64和 aarch32 转换的开销。

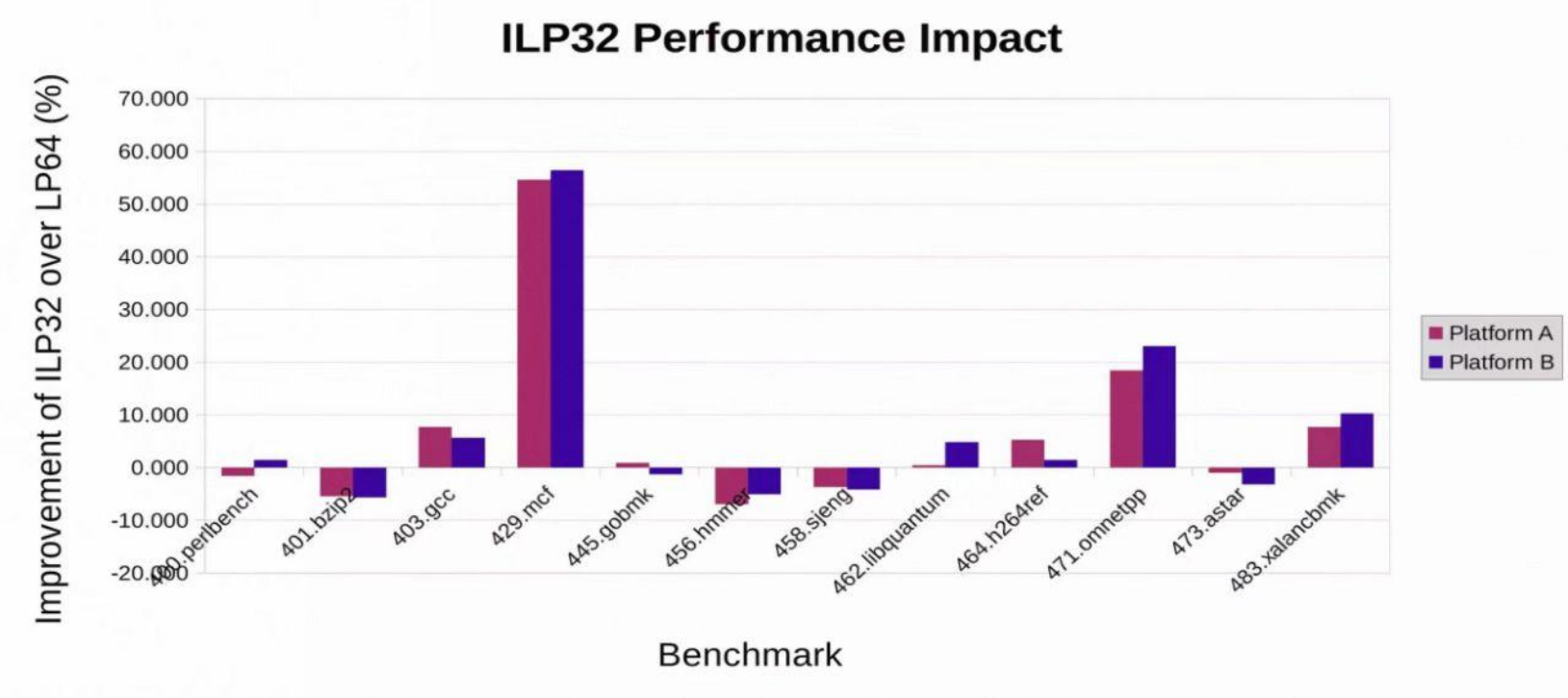
mips 推出n32 ABI的主要原因是mips 在九十年代被SGI（硅图）公司收购后，SGI 一方面推出了n64 ABI转向64位机器，但又舍不得放弃原来mips o32 ABI上开发的软件。

ILP32 On 64bit

Performance Data

- On Core i7 2600K 3.40GHz:
  - Improved SPEC CPU 2K/2006 INT geomean by 7-10% over ia32 and 5-8% over Intel64.
  - Improved SPEC CPU 2K/2006 FP geomean by 5-11% over ia32.
  - Very little changes in SPEC CPU 2K/2006 FP geomean, comparing against Intel64.
  - Comparing against ia32 PIC, x32 PIC:
    - Improved SPEC CPU 2K INT by another 10%.
    - Improved SPEC CPU 2K FP by another 3%.
    - Improved SPEC CPU 2006 INT by another 6%
    - Improved SPEC CPU 2006 FP by another 2%.

ILP32 On 64bit



Run ILP32 on RV64

由第一部分内容可知，若要在 RV64 上运行遵循ILP32 ABI的程序，一方面我们需要在工具链上提供对ILP32的支持，另一方面，我们需要在操作系统上提供这方面的支持。

由第二部分内容，我们知晓了要在 RV64 位架构上支持ILP32 ABI大概需要实现什么功能。

Run ILP32 on RV64

1. 当使用-march=rv64* -mabi=ilp32* 时，Data Representation需要设置int、long、pointer为32.
2. 做load、store时，对地址进行零扩展还是符号位扩展？
3. 生成的二进制文件应该是 ELF32 还是 ELF64 ？
4. 怎么区分生成的二进制文件与原生的rv32 ilp32或rv64 lp64
5. C 库中怎么添加支持？
6. kernel（略）

Run ILP32 on RV64

1.Data Representation：

GCC ：gcc/config/riscv/riscv.h

```
/* Set the sizes of the core types.  */
#define SHORT_TYPE_SIZE 16
#define INT_TYPE_SIZE 32
#define LONG_LONG_TYPE_SIZE 64
#define POINTER_SIZE (riscv_abi >= ABI_LP64 ? 64 : 32)
#define LONG_TYPE_SIZE POINTER_SIZE

#define FLOAT_TYPE_SIZE 32
#define DOUBLE_TYPE_SIZE 64
#define LONG_DOUBLE_TYPE_SIZE 128
```

Run I

```
class LLVM_LIBRARY_VISIBILITY RISCV64TargetInfo : public RISCVTargetInfo {
public:
  RISCV64TargetInfo(const llvm::Triple &Triple, const TargetOptions &Opts)
      : RISCVTargetInfo(Triple, Opts) {
    LongWidth = LongAlign = PointerWidth = PointerAlign = 64;
    IntMaxType = Int64Type = SignedLong;
    resetDataLayout("e-m:e-p:64:64-i64:64-i128:128-n32:64-S128");
  }

  bool setABI(const std::string &Name) override {
    if (Name == "lp64" || Name == "lp64f" || Name == "lp64d") {
      ABI = Name;
      return true;
    }

    if (Name == "ilp32") {
      ABI = Name;
      resetDataLayout("e-m:e-p:32:32-i64:64-n32:64-S128");
      LongWidth = LongAlign = PointerWidth = PointerAlign = 32;
      IntMaxType = SignedLongLong;
      return true;
    }
    return false;
  }
```

Run ILP32 on RV64

2.符号位扩展还是零扩展？

经过分析，发现rv64的指令中，addi、sub、mul sll等指令都有带符号位扩展的版本，而且也有32位数据的load、store指令lw、sw。除此之外，符号位扩展到64位有特定的指令sext.w，而零扩展到64位则需要两条左移右移指令组合才能完成。

Run ILP32 on RV64

3、4 参考x86-64 x32和mips n32，以及aarch64 ilp32，应该生成32位的ELF文件。但为了和rv32 ilp32生成的32位ELF文件区分，我们在ELF文件上增加了一个FLAG：X32。这部分工作，GCC需要在binutils中完成，相对比较繁琐。LLVM只需

*/MCTargetDesc/RISCVAsmBackend.h

完成对应内容即可

谢 谢

欢迎交流合作

2023/09/23