NASM – THE NETWIDE ASSEMBLER
网络汇编程序

version 2.14.03rc2
版本 2.14.03 rc2

Contents
目录

10

12

13

C.2.3 版本 0.98.37。 . . . . .2.4 版本
0.98.36。 . . . . .2.5 版本

14
14

16

Chapter 1: Introduction
第一章: 导言

1.1 What Is NASM?
1.1 什么是 NASM?

The Netwide Assembler, NASM, is an 80x86 and x86−64 assembler designed for portability and modularity. It supports a range of object file formats, including Linux and `*BSD a.out`, `ELF`, `COFF`, `Mach−O`, 16−bit and 32−bit `OBJ` (OMF) format, `Win32` and `Win64`. It will also output plain binary files, Intel hex and Motorola S−Record formats. Its syntax is designed to be simple and easy to understand, similar to the syntax in the Intel Software Developer Manual with minimal complexity. It supports all currently known x86 architectural extensions, and has strong support for macros.
Netwide 汇编程序 NASM 是一个 80x86 和 x86-64 的汇编程序，用于可移植性和模块化。它支持各种对象文件格式，包括 Linux 和 * BSD a.out、 ELF、 COFF、 Mach-o、16 位和 32 位 OBJ (OMF)格式、win32 和 Win64。它还可以输出普通的二进制文件，Intel 十六进制和 Motorola s-Record 格式。它的语法设计简单易懂，类似于英特尔软件开发者手册中的语法，只有最少的复杂性。它支持所有目前已知的 x86 架构扩展，并对宏有强大的支持。

NASM also comes with a set of utilities for handling the `RDOFF` custom object−file format.
NASM 还提供了一系列处理 RDOFF 自定义对象文件格式的工具。

1.1.1 License Conditions
1.1.1 许可证条件

Please see the file `LICENSE`, supplied as part of any NASM distribution archive, for the license conditions under which you may use NASM. NASM is now under the so−called 2−clause BSD license, also known as the simplified BSD license.
请参阅作为 NASM 分发档案的一部分提供的文件 LICENSE，了解您可以使用 NASM 的许可条件。NASM 现在是所谓的 2 条款 BSD 许可证，也被称为简化 BSD 许可证。

Copyright 1996−2017 the NASM Authors – All rights reserved.
版权 1996-2017 NASM 作者-保留所有权利。

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
在满足以下条件的情况下，允许以源代码和二进制形式重新分配和使用，无论是否进行修改:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

源代码的重新发布必须保留上述版权声明，此条件列表和以下免责声明。

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

以二进制形式重新发行必须在发行所提供的文件和/或其他材料中复制上述版权声明、本条件清单和以下免责声明。

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
本软件由版权持有人及撰稿人提供，并不接受任何明示或默示的保证，包括但不限于默示的适销性及适用于特定用途的保证。在任何情况下，版权所有人或贡献者均不应对任何直接、间接、附带、特殊、示范或间接损害赔偿(包括

17

18
18 岁

Chapter 2: Running NASM
第二章: 运行 NASM

2.1 NASM Command−Line Syntax
2.1 NASM 命令行语法

To assemble a file, you issue a command of the form
要组装一个文件，你需要发出一个命令

```
nasm -f <format> <filename> [-o <output>]
Nasm-f < format > < filename > [-o < output > ]
```

For example,
例如,

```
nasm -f elf myfile.asm
Nasm-f elf myfile.asm
```

will assemble `myfile.asm` into an `ELF` object file `myfile.o`.

And `nasm -f bin myfile.asm -o myfile.com`

将 myfile.asm 组装成一个 ELF 对象文件 myfile.o。和 nasm-f bin

myfile.asm-o myfile. com

will assemble `myfile.asm` into a raw binary file `myfile.com`.
会将 myfile.asm 组装成一个原始二进制文件 myfile.com。

To produce a listing file, with the hex codes output from NASM displayed on the left of the original sources, use the `-l` option to give a listing file name, for example:
要生成一个列表文件，NASM 输出的十六进制代码显示在原始源的左侧，可以使用 -l 选项给出一个列表文件名，例如:

```
nasm -f coff myfile.asm -l myfile.lst
我的文件，我的文件
```

To get further usage instructions from NASM, try typing
要从 NASM 获得更多的使用说明，试着输入

```
nasm -h
Nasm-h
```

The option `--help` is an alias for the `-h` option.
Help 选项是 -h 选项的别名。

The option `-hf` will also list the available output file formats, and what they are.
选项 -hf 还将列出可用的输出文件格式，以及它们是什么。

If you use Linux but aren't sure whether your system is `a.out` or `ELF`, type
如果你使用 Linux 但是不确定你的系统是 a.out 还是 ELF，输入

```
file nasm
文件 nasm
```

(in the directory in which you put the NASM binary when you installed it). If it says something like
(在您安装 NASM 二进制文件时所在的目录中)

```
nasm: ELF 32-bit LSB executable i386 (386 and up) Version 1
Nasm: ELF 32 位 LSB 可执行文件 i386(386 及以上版本) Version 1
```

then your system is `ELF`, and you should use the option `-f elf` when you want NASM to produce Linux object files. If it says

那么你的系统就是 ELF，当你想要 NASM 生成 Linux 对象文件时，你应该使用 -f ELF 选项。如果它说

```
nasm: Linux/i386 demand-paged executable (QMAGIC)
Nasm: Linux/i386 需求分页可执行文件(QMAGIC)
```

or something similar, your system is `a.out`, and you should use `-f aout` instead (Linux `a.out` systems have long been obsolete, and are rare these days.)
或者类似的东西，您的系统是 a.out，您应该使用 -f aout (Linux a.out 系统已经过时很久了，现在已经很少见了)

Like Unix compilers and assemblers, NASM is silent unless it goes wrong: you won't see any output at all, unless it gives error messages.
像 Unix 编译器和汇编器一样，NASM 是静默的，除非出错: 除非它提供错误消息，否则根本看不到任何输出。

2.1.1 The `-o` Option: Specifying the Output File Name
2.1.1-o 选项: 指定输出文件名

NASM will normally choose the name of your output file for you; precisely how it does this is dependent on the object file format. For Microsoft object file formats (`obj`, `win32` and `win64`), it will remove the
NASM 通常会为您选择输出文件的名称; 确切地说，它是如何做到这一点取决于对象文件格式。对于 Microsoft 对象文件格式(obj，win32 和 win64)，它会删除

`.asm` extension (or whatever extension you like to use – NASM doesn't care) from your source file name and substitute `.obj`. For Unix object file formats (`aout`, `as86`, `coff`, `elf32`, `elf64`, `elfx32`, `ieee`, `macho32` and `macho64`) it will substitute `.o`. For `dbg`, `rdf`, `ith` and `srec`, it will use `.dbg`, `.rdf`,
.Asm 扩展名(或者任何你喜欢使用的扩展名— NASM 不在乎)从你的源文件名中删除，代之以。对象。对于 Unix 对象文件格式(大约 as86，coff，elf32，elf64，elfx32，ieee，macho32 和 macho64)，它将替代。对于 dbg，rdf，ith 和 srec，它将使用。Dbg，.Rdf，
`.ith` and `.srec`, respectively, and for the `bin` format it will simply remove the extension, so that `myfile.asm` produces the output file `myfile`.
.还有。对于 bin 格式，它只需删除扩展名，这样 myfile.asm 就会生成输出文件 myfile。

If the output file already exists, NASM will overwrite it, unless it has the same name as the input file, in which case it will give a warning and use `nasm.out` as the output file name instead.
如果输出文件已经存在，NASM 将覆盖它，除非它与输入文件具有相同的名称，在这种情况下，它将发出警告并使用 NASM.out 作为输出文件名。

For situations in which this behaviour is unacceptable, NASM provides the `-o` command−line option, which allows you to specify your desired output file name. You invoke `-o` by following it with the name you wish for the output file, either with or without an intervening space. For example:
对于这种行为是不可接受的情况，NASM 提供了-o 命令行选项，允许您指定您想要的输出文件名。你可以通过在 -o 后面加上你希望输出文件的名称来调用它，无论是否加入空格。例如:

```
nasm -f bin program.asm -o program.com
Nasm-f bin program. asm-o program
nasm -f bin driver.asm -odriver.sys
Nasm-f bin driver.asm-odriver.sys
```

Note that this is a small o, and is different from a capital O , which is used to specify the number of optimisation passes required. See section 2.1.23.
请注意，这是一个小的 o，不同于大写的 o，后者用于指定所需的优化通过次数。参见第 2.1.23 节。

2.1.2 The `-f` Option: Specifying the Output File Format
2.1.2-f 选项: 指定输出文件格式

If you do not supply the `-f` option to NASM, it will choose an output file format for you itself. In the distribution versions of NASM, the default is always `bin`; if you've compiled your own copy of NASM, you can redefine `OF_DEFAULT` at compile time and choose what you want the default to be.
如果您不向 NASM 提供 -f 选项，它将为您自己选择输出文件格式。在 NASM 的发行版本中，默认值始终是 bin; 如果您已经编译了自己的 NASM 副本，则可以在编译时重新定义 of_default 并选择您希望的默认值。

Like `-o`, the intervening space between `-f` and the output file format is optional; so `-f elf` and `-felf` are both valid.
和 -o 一样，-f 和输出文件格式之间的空格是可选的，所以 -f elf 和 -felf 都是有效的。

A complete list of the available output file formats can be given by issuing the command `nasm -hf`.
可以通过发出命令 nasm-hf 给出可用输出文件格式的完整列表。

2.1.3 The `-l` Option: Generating a Listing File
2.1.3-l 选项: 生成一个列表文件

If you supply the `-l` option to NASM, followed (with the usual optional space) by a file name, NASM will generate a source−listing file for you, in which addresses and generated code are listed on the left, and the actual source code, with expansions of multi−line macros (except those which specifically request no expansion in source listings: see section 4.3.11) on the right. For example:
如果您向 NASM 提供 -l 选项，后面跟着一个文件名(通常使用可选空格)，NASM 将为您生成一个源代码列表文件，其中地址和生成的代码在左边列出，实际的源代码在右边列出多行宏的扩展(除了那些在源代码列表中明确要求不进行扩展的文件: 参见第 4.3.11 节)。例如:

```
nasm -f elf myfile.asm -l myfile.lst
Nasm-f elf myfile.asm-l myfile.lst
```

If a list file is selected, you may turn off listing for a section of your source with `[list -]`, and turn it back on with `[list +]`, (the default, obviously). There is no "user form" (without the brackets). This can be used to list only sections of interest, avoiding excessively long listings.
如果选择了一个列表文件，您可以用[list-]关闭源的某个部分的列表，然后用[list +](显然是默认的)重新打开它。没有“用户表单”(没有括号)。这只能用于列出感兴趣的部分，避免过长的列表。

2.1.4 The `-M` Option: Generate Makefile Dependencies
2.1.4-m 选项: 生成 Makefile 依赖项

This option can be used to generate makefile dependencies on stdout. This can be redirected to a file for further processing. For example:
此选项可用于生成标准输出上的 makefile 依赖项。它可以被重定向到一个文件进行进一步的处理。例如:

```
nasm -M myfile.asm > myfile.dep
Nasm-m myfile.asm > myfile.dep
```

2.1.5 The `-MG` Option: Generate Makefile Dependencies
2.1.5-MG 选项: Generate Makefile Dependencies

This option can be used to generate makefile dependencies on stdout. This differs from the `-M` option in that if a nonexisting file is encountered, it is assumed to be a generated file and is added to the dependency list without a prefix.
这个选项可以用来在 stdout 上生成 makefile 依赖项。这与 -m 选项的不同之处在于，如果遇到一个不存在的文件，则假定它是一个生成的文件，并将其添加到依赖项列表中，而不带前缀。

2.1.6 The `-MF` Option: Set Makefile Dependency File
2.1.6-MF 选项: Set Makefile Dependency File

This option can be used with the `-M` or `-MG` options to send the output to a file, rather than to stdout.
这个选项可以和 -m 或 -MG 选项一起使用，将输出发送到一个文件，而不是发送到标准输出。
For example:
例如:

```
nasm -M -MF myfile.dep myfile.asm
Nasm-m-MF myfile.dep myfile.asm
```

2.1.7 The `-MD` Option: Assemble and Generate Dependencies
2.1.7-MD 选项: 汇编和生成依赖项

The `-MD` option acts as the combination of the `-M` and `-MF` options (i.e. a filename has to be specified.)
MD 选项充当 -m 和 -MF 选项的组合(即必须指定文件名)
However, unlike the `-M` or `-MG` options, `-MD` does *not* inhibit the normal operation of the assembler.
然而，与 -m 或 -MG 选项不同，-MD 不会抑制汇编程序的正常运行。
Use this to automatically generate updated dependencies with every assembly session. For example:
使用它可以自动生成每个装配会话的更新依赖项。例如:

```
nasm -f elf -o myfile.o -MD myfile.dep myfile.asm
Nasm-f elf-o myfile.o-MD myfile.dep myfile.asm
```

If the argument after `-MD` is an option rather than a filename, then the output filename is the first applicable one of:
如果 -MD 后面的参数是一个选项而不是一个文件名，那么输出文件名是第一个适用的参数:

• the filename set in the `-MF` option;
在 -MF 选项中设置的文件名;

• the output filename from the `-o` option with `.d` appended;
来自 -o 选项的输出文件名，后面附加.d;

• the input filename with the extension set to `.d`.
扩展名设置为.d 的输入文件名。

2.1.8 The –`MT` Option: Dependency Target Name
2.1.8-MT 选项: Dependency Target Name

The `-MT` option can be used to override the default name of the dependency target. This is normally the same as the output filename, specified by the `-o` option.
MT 选项可以用来覆盖依赖目标的默认名称。这通常与输出文件名相同，由 -o 选项指定。

2.1.9 The –`MQ` Option: Dependency Target Name (Quoted)
2.1.9-MQ 选项: Dependency Target Name (引用)

The `-MQ` option acts as the `-MT` option, except it tries to quote characters that have special meaning in Makefile syntax. This is not foolproof, as not all characters with special meaning are quotable in Make. The default output (if no `-MT` or `-MQ` option is specified) is automatically quoted.
MQ 选项充当 -MT 选项，除非它试图引用在 Makefile 语法中有特殊含义的字符。这并不是万无一失的，因为并不是所有具有特殊含义的字符都可以在 Make 中引用。默认输出(如果没有指定 -MT 或 -MQ 选项)被自动引用。

2.1.10 The –`MP` Option: Emit phony targets
2.1.10-MP 选项: 发射假目标

When used with any of the dependency generation options, the `-MP` option causes NASM to emit a phony target without dependencies for each header file. This prevents Make from complaining if a header file has been removed.
当与任何依赖生成选项一起使用时,-MP 选项会导致 NASM 为每个头文件发出一个没有依赖的虚假目标。这样可以防止 Make 在头文件被删除时抱怨。

2.1.11 The –`MW` Option: Watcom Make quoting style
2.1.11-MW 选项: Watcom Make 报价风格

This option causes NASM to attempt to quote dependencies according to Watcom Make conventions rather than POSIX Make conventions (also used by most other Make variants.) This quotes `#` as `$#` rather than `\#`, uses `&` rather than `\` for continuation lines, and encloses filenames containing whitespace in double quotes.
这个选项会导致 NASM 尝试根据 Watcom Make 约定而不是 POSIX Make 约定来引用依赖项(也被大多数其他 Make 变体使用)这个引号 # 作为 $# 而不是 #，使用 & 而不是作为延续行，并将包含空格的文件名用双引号括起来。

2.1.12 The –`F` Option: Selecting a Debug Information Format
2.1.12-f 选项: 选择调试信息格式

This option is used to select the format of the debug information emitted into the output file, to be used by a debugger (or *will* be). Prior to version 2.03.01, the use of this switch did *not* enable output of the selected debug info format. Use `-g`, see section 2.1.13, to enable output. Versions 2.03.01 and later automatically enable `-g` if `-F` is specified.
这个选项用于选择发送到输出文件的调试信息的格式，供调试器使用(或将要使用)。在版本 2.03.01 之前，这个开关的使用不允许输出所选的调试信息格式。Use-g，参见 2.1.13 节，启用输出。Version2.03.01 和更高版本如果指定了 -f，则自动启用 -g。

A complete list of the available debug file formats for an output format can be seen by issuing the command `nasm -f <format> -y`. Not all output formats currently support debugging output. See section 2.1.27.
通过发出命令 nasm-f < format >-y，可以看到输出格式的可用调试文件格式的完整列表。参见第 2.1.27 节。

This should not be confused with the `-f dbg` output format option, see section 7.14.
这不应该与 -f dbg 输出格式选项混淆，参见第 7.14 节。

2.1.13 The `-g` Option: Enabling Debug Information.
2.1.13-g 选项: 启用调试信息。

This option can be used to generate debugging information in the specified format. See section 2.1.12. Using `-g` without `-F` results in emitting debug info in the default format, if any, for the selected output format. If no debug information is currently implemented in the selected output format, `-g` is *silently ignored*.
这个选项可以用来生成指定格式的调试信息。参见第 2.1.12 节。使用 -g 而不使用 -f 会导致以所选输出格式的默认格式(如果有的话)发出调试信息。如果当前没有在选定的输出格式中实现调试信息,-g 将被默认忽略。

2.1.14 The `-X` Option: Selecting an Error Reporting Format
2.1.14-x 选项: 选择错误报告格式

This option can be used to select an error reporting format for any error messages that might be produced by NASM.
这个选项可以用来为 NASM 可能产生的任何错误信息选择一个错误报告格式。

Currently, two error reporting formats may be selected. They are the `-Xvc` option and the `-Xgnu` option. The GNU format is the default and looks like this:
目前，可以选择两种错误报告格式。它们是 -Xvc 选项和 -Xgnu 选项。GNU 格式是默认的，看起来像这样:

```
filename.asm:65: error: specific error message
文件名.asm: 65: error: specific error message
```

where `filename.asm` is the name of the source file in which the error was detected, `65` is the source file line number on which the error was detected, `error` is the severity of the error (this could be `warning`), and `specific error message` is a more detailed text message which should help pinpoint the exact problem.
其中 filename.asm 是检测到错误的源文件的名称，65 是检测到错误的源文件行号，错误是错误的严重程度(这可能是警告)，具体的错误消息是更详细的文本消息，应该有助于查明确切的问题。

The other format, specified by `-Xvc` is the style used by Microsoft Visual C++ and some other programs.
- Xvc 指定的另一种格式是 microsoftvisualc + + 和其他一些程序使用的样式。
It looks like this:
它看起来像这样:

```
filename.asm(65) : error: specific error message
Asm (65) : error: 特定的错误消息
```

where the only difference is that the line number is in parentheses instead of being delimited by colons.
唯一的区别是行号在括号里，而不是用冒号分隔。

See also the `Visual C++` output format, section 7.5.
另见 visualc + + 输出格式，第 7.5 节。

2.1.15 The `-Z` Option: Send Errors to a File
2.1.15-z 选项: 向文件发送错误

Under `MS-DOS` it can be difficult (though there are ways) to redirect the standard−error output of a program to a file. Since NASM usually produces its warning and error messages on `stderr`, this can make it hard to capture the errors if (for example) you want to load them into an editor.
在 MS-DOS 下，将程序的标准错误输出重定向到文件是很困难的(尽管有一些方法)。由于 NASM 通常在 stderr 上生成警告和错误消息，因此如果(例如)您希望将错误加载到编辑器中，就很难捕获错误。

NASM therefore provides the `-Z` option, taking a filename argument which causes errors to be sent to the specified files rather than standard error. Therefore you can redirect the errors into a file by typing
NASM 因此提供了 -z 选项，使用一个文件名参数，这个参数会导致错误发送到指定的文件，而不是标准的错误。因此，你可以通过输入

```
nasm -Z myfile.err -f obj myfile.asm
Nasm-z myfile.err-f obj myfile.asm
```

In earlier versions of NASM, this option was called `-E`, but it was changed since `-E` is an option conventionally used for preprocessing only, with disastrous results. See section 2.1.21.
在 NASM 的早期版本中，这个选项被称为 -e，但是它被更改了，因为 -e 通常只用于预处理，结果是灾难性的。参见第 2.1.21 节。

2.1.16 The `-s` Option: Send Errors to `stdout`
2.1.16 s 选项: 将错误发送到 stdout

The `-s` option redirects error messages to `stdout` rather than `stderr`, so it can be redirected under
S 选项将错误信息重定向到 stdout 而不是 stderr，因此可以在
`MS-DOS`. To assemble the file `myfile.asm` and pipe its output to the `more` program, you can type:
要组装文件 `myfile.asm` 并将其输出管道到 `more` 程序，可以输入：

```
nasm -s -f obj myfile.asm | more
Nasm-s-f obj myfile.asm | more
```

See also the `-Z` option, section 2.1.15.
另见 -z 选项，第 2.1.15 节。

2.1.17 The `-i` Option: Include File Search Directories
2.1.17-i 选项: 包含文件搜索目录

When NASM sees the `%include` or `%pathsearch` directive in a source file (see section 4.6.1, section 4.6.2 or section 3.2.3), it will search for the given file not only in the current directory, but

also in any directories specified on the command line by the use of the `-i` option. Therefore you can include files from a macro library, for example, by typing
当 NASM 在源文件中看到% include 或% pathsearch 指令时(参见第 4.6.1 节、第 4.6.2 节或第 3.2.3 节)，它将不仅在当前目录中搜索给定的文件，而且通过使用 -i 选项在命令行上指定的任何目录中搜索给定的文件。因此，你可以从宏库中包含文件，例如，输入

```
nasm -ic:\macrolib\ -f obj myfile.asm
Nasm-ic: macrolib-f obj myfile.asm
```

(As usual, a space between `-i` and the path name is allowed, and optional).
(通常，允许在 -i 和路径名之间留一个空格，并且是可选的)。

Prior NASM 2.14 a path provided in the option has been considered as a verbatim copy and providing a path separator been up to a caller. One could implicitly concatenate a search path together with a filename. Still this was rather a trick than something useful. Now the trailing path separator is made to always present, thus `-ifoo` will be considered as the `-ifoo/` directory.
Prior NASM 2.14 在选项中提供的路径被认为是一个逐字拷贝，并提供一个路径分隔符由调用者决定。一个人可以隐式连接一个搜索路径和一个文件名。尽管如此，这仍然是一个技巧，而不是有用的东西。现在尾随路径分隔符被设置为始终显示，因此-ifoo 将被视为-ifoo/目录。

If you want to define a *standard* include search path, similar to `/usr/include` on Unix systems, you should place one or more `-i` directives in the `NASMENV` environment variable (see section 2.1.34).
如果您想定义一个标准的 include 搜索路径，类似于 Unix 系统上的/usr/include，您应该在 NASMENV 环境变量中放置一个或多个 -i 指令(参见 2.1.34 节)。

For Makefile compatibility with many C compilers, this option can also be specified as `-I`.
为了使 Makefile 与许多 c 编译器兼容，这个选项也可以指定为 -i。

2.1.18 The −p Option: Pre−Include a File
2.1.18-p 选项: Pre-Include a File

NASM allows you to specify files to be *pre−included* into your source file, by the use of the `-p` option. So running
NASM 允许你使用 -p 选项指定预包含到源文件中的文件。所以运行

```
nasm myfile.asm -p myinc.inc
Nasm myfile.asm-p myinc
```

is equivalent to running `nasm myfile.asm` and placing the directive `%include "myinc.inc"` at the start of the file.
相当于运行 nasm myfile.asm 并将指令% include“ myinc.inc”放在文件的开头。

`--include` option is also accepted.
`--include` 选项也被接受。

For consistency with the `-I`, `-D` and `-U` options, this option can also be specified as `-P`.
为了与 -i、-d 和 -u 选项保持一致，还可以将该选项指定为 -p。

2.1.19 The `-d` Option: Pre−Define a Macro
2.1.19-d 选项: Pre-Define a Macro

Just as the `-p` option gives an alternative to placing `%include` directives at the start of a source file, the `-d` option gives an alternative to placing a `%define` directive. You could code
正如-p 选项提供了在源文件开头放置% include 指令的替代方法一样,-d 选项提供了放置% define 指令的替代方法。你可以编写代码

```
nasm myfile.asm -dFOO=100
Nasm myfile.asm-dFOO = 100
```

as an alternative to placing the directive
作为指令的替代方案

```
%define FOO 100
% 定义 FOO 100
```

at the start of the file. You can miss off the macro value, as well: the option `-dFOO` is equivalent to coding `%define FOO`. This form of the directive may be useful for selecting assembly−time options which are then tested using `%ifdef`, for example `-dDEBUG`.
在文件的开头。你也可以忽略宏值: 选项-dFOO 相当于编码% define FOO。这种形式的指令可能对选择汇编时选项很有用，然后使用% ifdef 进行测试，例如 -dDEBUG。

For Makefile compatibility with many C compilers, this option can also be specified as `-D`.
为了使 Makefile 与许多 c 编译器兼容，这个选项也可以指定为 -d。

2.1.20 The `-u` Option: Undefine a Macro
2.1.20-u 选项: 取消定义宏

The `-u` option undefines a macro that would otherwise have been pre−defined, either automatically or by a `-p` or `-d` option specified earlier on the command lines.
U 选项取消定义一个宏，否则这个宏将被预先定义，或者自动定义，或者通过命令行前面指定的 -p 或 -d 选项定义。

For example, the following command line:
例如，下面的命令行:

```
nasm myfile.asm -dFOO=100 -uFOO
Nasm myfile.asm-dFOO = 100-uFOO
```

would result in `FOO` *not* being a predefined macro in the program. This is useful to override options specified at a different point in a Makefile.
将导致 FOO 不是程序中预定义的宏。这对于重写 Makefile 中不同点指定的选项很有用。

For Makefile compatibility with many C compilers, this option can also be specified as `-U`.
为了使 Makefile 与许多 c 编译器兼容，这个选项也可以指定为 -u。

2.1.21 The `-E` Option: Preprocess Only
2.1.21-e 选项: 只适用于预处理

NASM allows the preprocessor to be run on its own, up to a point. Using the `-E` option (which requires no arguments) will cause NASM to preprocess its input file, expand all the macro references, remove all the comments and preprocessor directives, and print the resulting file on standard output (or save it to a file, if the `-o` option is also used).

NASM 允许预处理器在一定程度上独立运行。使用 -e 选项(不需要参数)将导致 NASM 预处理其输入文件，展开所有宏引用，删除所有注释和预处理器指令，并在标准输出中打印结果文件(如果也使用 -o 选项，则将其保存到文件中)。

This option cannot be applied to programs which require the preprocessor to evaluate expressions which depend on the values of symbols: so code such as
此选项不能应用于需要预处理器计算依赖于符号值的表达式的程序

```
%assign tablesize ($-tablestart)
% 赋予表大小($- tablestart)
```

will cause an error in preprocess−only mode.
将在仅预处理模式中引起错误。

For compatiblity with older version of NASM, this option can also be written −e. −E in older versions of NASM was the equivalent of the current −Z option, section 2.1.15.
为了与旧版本的 NASM 兼容，这个选项也可以在旧版本的 NASM 中写入 -e-e，相当于当前的 -z 选项，第 2.1.15 节。

2.1.22 The −a Option: Don’t Preprocess At All
2.1.22 a 选项: 根本不要预处理

If NASM is being used as the back end to a compiler, it might be desirable to suppress preprocessing completely and assume the compiler has already done it, to save time and increase compilation speeds. The −a option, requiring no argument, instructs NASM to replace its powerful preprocessor with a stub preprocessor which does nothing.
如果 NASM 被用作编译器的后端，那么可能需要完全禁止预处理并假设编译器已经这样做了，以节省时间并提高编译速度。A 选项，不需要参数，指示 NASM 用一个不做任何事情的存根预处理器替换它强大的预处理器。

2.1.23 The `-O` Option: Specifying Multipass Optimization
2.1.23-o 选项: 指定多路优化

Using the `-O` option, you can tell NASM to carry out different levels of optimization. Multiple flags can be specified after the `-O` options, some of which can be combined in a single option, e.g. `-Oxv`.
使用 -o 选项，您可以告诉 NASM 执行不同级别的优化。在 -o 选项之后可以指定多个标志，其中一些可以组合成单个选项，例如 -Oxv。

- `-O0`: No optimization. All operands take their long forms, if a short form is not specified, except conditional jumps. This is intended to match NASM 0.98 behavior.

O0: 没有优化。除了条件跳转之外，所有的操作数都采用长形式，如果没有指定短形式。这是为了匹配 NASM 0.98 的行为。

- `-O1`: Minimal optimization. As above, but immediate operands which will fit in a signed byte are optimized, unless the long form is specified. Conditional jumps default to the long form unless otherwise specified.

最小优化。如上所述，但是可以适应有符号字节的直接操作数是优化的，除非指定长形式。条件跳转默认为长格式，除非另有说明。

- `-Ox` (where `x` is the actual letter `x`): Multipass optimization. Minimize branch offsets and signed immediate bytes, overriding size specification unless the `strict` keyword has been used (see section 3.7). For compatibility with earlier releases, the letter `x` may also be any number greater than one. This number has no effect on the actual number of passes.

Ox (其中 x 是实际的字母 x) : Multipass 优化。最小化分支偏移量和有符号的即时字节，覆盖大小规范，除非使用了严格的关键字(见第 3.7 节)。为了与早期版本兼容，字母 x 也可以是任何大于 1 的数字。这个数字对实际的通过次数没有影响。

- `-Ov`: At the end of assembly, print the number of passes actually executed.

- Ov: 在装配结束时，打印实际执行的通行次数。

The `-Ox` mode is recommended for most uses, and is the default since NASM 2.09.
大多数情况下推荐使用 -Ox 模式，这是 NASM 2.09 以来的默认模式。

Note that this is a capital `O`, and is different from a small `o`, which is used to specify the output file name. See section 2.1.1.
注意，这是一个大写的 o，不同于用于指定输出文件名的小写的 o。参见 2.1.1 节。

2.1.24 The `-t` Option: Enable TASM Compatibility Mode
2.1.24-t 选项: 启用 TASM 兼容模式

NASM includes a limited form of compatibility with Borland's `TASM`. When NASM's `-t` option is used, the following changes are made:
NASM 包括一个与 Borland 的 TASM 兼容的有限形式。当使用 NASM 的 -t 选项时，会进行以下更改:

- local labels may be prefixed with `@@` instead of `.`

本地标签可以前缀@@，而不是。

- size override is supported within brackets. In TASM compatible mode, a size override inside square brackets changes the size of the operand, and not the address type of the operand as it does in NASM syntax. E.g. `mov eax,[DWORD val]` is valid syntax in TASM compatibility mode. Note that you lose the ability to override the default address type for the instruction.

括号内支持大小重写。在 TASM 兼容模式下，方括号内的大小重写会改变操作数的大小，而不是像 NASM 语法中那样改变操作数的地址类型。例如 mov eax，[DWORD val]在 TASM 兼容模式下是有效的语法。注意，你失去了覆盖指令的默认地址类型的能力。

- unprefixed forms of some directives supported (`arg`, `elif`, `else`, `endif`, `if`, `ifdef`, `ifdifi`, `ifndef`, `include`, `local`)

支持某些指令的无前缀形式(arg，elif，else，endif，if，ifdef，ifdifi，ifndef，include，local)

2.1.25 The `-w` and `-W` Options: Enable or Disable Assembly Warnings
2.1.25-w 和-w 选项: 启用或禁用程序集警告

NASM can observe many conditions during the course of assembly which are worth mentioning to the user, but not a sufficiently severe error to justify NASM refusing to generate an output file. These conditions are reported like errors, but come up with the word 'warning' before the message. Warnings do not prevent NASM from generating an output file and returning a success status to the operating system.
NASM 可以在装配过程中观察到许多值得提醒用户的情况，但是没有足够严重的错误来证明 NASM 拒绝生成输出文件。这些条件会像错误一样被报告，但是在消息之前会出现"警告"这个词。警告不会阻止 NASM 生成输出文件并返回操作系统的成功状态。

Some conditions are even less severe than that: they are only sometimes worth mentioning to the user. Therefore NASM supports the `-w` command−line option, which enables or disables certain classes of assembly warning. Such warning classes are described by a name, for example `orphan-labels`; you can enable warnings of this class by the command−line option `-w+orphan-labels` and disable it by `-w-orphan-labels`.
有些情况甚至没有那么严重: 它们只是有时候值得向用户提及。因此，NASM 支持 -w 命令行选项，该选项启用或禁用某些类的汇编警告。这样的警告类通过名称来描述，例如孤儿标签; 您可以通过命令行选项 -w + 孤儿标签来启用这个类的警告，并通过 -w-孤儿标签来禁用它。

The current warning classes are:
当前的警告类是:

• `other` specifies any warning not otherwise specified in any class. Enabled by default.
其他指定任何未在任何类中指定的警告。默认情况下启用。

• `macro-params` covers warnings about multi−line macros being invoked with the wrong number of parameters. Enabled by default; see section 4.3.1 for an example of why you might want to disable it.
Macro-params 覆盖了关于多行宏被错误数量的参数调用的警告。默认情况下启用; 参见第 4.3.1 节的例子来说明为什么你可能想要禁用它。

• `macro-selfref` warns if a macro references itself. Disabled by default.
如果宏引用自己，宏自由会发出警告。默认情况下禁用。

- macro-defaults warns when a macro has more default parameters than optional parameters. Enabled by default; see section 4.3.5 for why you might want to disable it.

当宏的默认参数多于可选参数时，宏默认值会发出警告。默认情况下启用； 参见第 4.3.5 节了解为什么你可能想要禁用它。

- orphan-labels covers warnings about source lines which contain no instruction but define a label without a trailing colon. NASM warns about this somewhat obscure condition by default; see section 3.1 for more information.

孤儿标签涵盖了关于源代码行的警告，这些源代码行不包含任何指令，但是定义了一个没有尾随冒号的标签。默认情况下，NASM 警告这个有点模糊的条件； 更多信息参见第 3.1 节。

- number-overflow covers warnings about numeric constants which don't fit in 64 bits. Enabled by default.

数字溢出覆盖了关于不适合 64 位的数字常量的警告。默认情况下启用。

- gnu-elf-extensions warns if 8−bit or 16−bit relocations are used in -f elf format. The GNU extensions allow this. Disabled by default.

Gnu-elf-extensions 警告 8 位或 16 位重定位是否使用 -f elf 格式。GNU 扩展允许这样做。默认情况下禁用。

- float-overflow warns about floating point overflow. Enabled by default.

浮点溢出警告浮点溢出。默认启用。

- float-denorm warns about floating point denormals. Disabled by default.

Float-denorm 警告浮点数。默认情况下禁用。

- float-underflow warns about floating point underflow. Disabled by default.

浮点下流警告浮点下流。默认情况下禁用。

- float-toolong warns about too many digits in floating−point numbers. Enabled by default.

Float-toolong 警告浮点数过多。默认启用。

- user controls %warning directives (see section 4.9). Enabled by default.

用户控件% 警告指令(见第 4.9 节)。默认启用。

- lock warns about LOCK prefixes on unlockable instructions. Enabled by default.

LOCK 警告解锁指令上的 LOCK 前缀。默认启用。

- hle warns about invalid use of the HLE XACQUIRE or XRELEASE prefixes. Enabled by default.

HLE 警告对 HLE XACQUIRE 或 XRELEASE 前缀的无效使用。默认情况下启用。

- bnd warns about ineffective use of the BND prefix when a relaxed form of jmp instruction becomes jmp short form. Enabled by default.

BND 警告当一个放松的 jmp 指令变成 jmp 缩写形式时，BND 前缀的使用效果不佳。默认启用。

- zext-reloc warns that a relocation has been zero−extended due to limitations in the output format. Enabled by default.

Zext-reloc 警告说，由于输出格式的限制，重定位已被零扩展。默认启用。

- ptr warns about keywords used in other assemblers that might indicate a mistake in the source code. Currently only the MASM PTR keyword is recognized. Enabled by default.

Ptr 警告在其他汇编程序中使用的关键字，这些关键字可能表明源代码中有错误。目前只有 MASM PTR 关键字被识别。默认启用。

- bad-pragma warns about a malformed or otherwise unparsable %pragma directive. Disabled by default.

Bad-pragma 警告格式不正确或无法解析的% pragma 指令。默认情况下禁用。

- unknown-pragma warns about an unknown %pragma directive. This is not yet implemented. Disabled by default.

Unknown-pragma 警告未知% pragma 指令。这还没有实现。默认为 Disabled。

- not-my-pragma warns about a %pragma directive which is not applicable to this particular assembly session. This is not yet implemented. Disabled by default.

Not-my-pragma 警告不适用于此特定程序集会话的% pragma 指令。这还没有实现。默认禁用。

- unknown-warning warns about a -w or -W option or a [WARNING] directive that contains an unknown warning name or is otherwise not possible to process. Disabled by default.

Unknown-WARNING 警告包含未知警告名称或无法处理的 -w 或 -w 选项或[WARNING]指令。默认禁用。

- all is an alias for *all* suppressible warning classes. Thus, -w+all enables all available warnings, and -w-all disables warnings entirely (since NASM 2.13).

All 是所有可抑制的警告类的别名。因此,-w + all 启用所有可用的警告,-w-all 完全禁用警告(从 NASM 2.13 开始)。

Since version 2.00, NASM has also supported the gcc–like syntax -Wwarning-class and -Wno-warning-class instead of -w+warning-class and -w-warning-class, respectively; both syntaxes work identically.
自从版本 2.00 以来，NASM 还分别支持类似 gcc 的语法 -Wwarning-class 和 -Wno-warning-class，而不是 -w + warning-class 和 -w-warning-class; 这两种语法的工作原理是一样的。

The option -w+error or -Werror can be used to treat warnings as errors. This can be controlled on a per warning class basis (-w+error=*warning−class* or -Werror=*warning−class*); if no *warning−class* is specified NASM treats it as -w+error=all; the same applies to -w-error or -Wno-error, of course.
选项 -w + error 或 -Werror 可用于将警告视为错误。这可以在每个警告类的基础上进行控制(- w + error = warning-class 或-Werror = warning-class) ; 如果没有指定警告类，NASM 将其视为 -w + error = all; 当然，-w-error 或 -Wno-error 也是如此。

In addition, you can control warnings in the source code itself, using the [WARNING] directive. See section 6.13.
另外，你可以使用[WARNING]指令在源代码中控制警告。参见第 6.13 节。

2.1.26 The −`v` Option: Display Version Info
2.1.26-v 选项: 显示版本信息

Typing `NASM −v` will display the version of NASM which you are using, and the date on which it was compiled.
键入 NASM-v 将显示您正在使用的 NASM 版本，以及编译它的日期。

You will need the version number if you report a bug.
如果你报告错误，你将需要版本号。

For command−line compatibility with Yasm, the form `−−v` is also accepted for this option starting in NASM version 2.11.05.
为了与 Yasm 的命令行兼容，从 NASM 版本 2.11.05 开始，也接受表单 -v 作为此选项。

2.1.27 The −`y` Option: Display Available Debug Info Formats
2.1.27-y 选项: 显示可用的调试信息格式

Typing `nasm −f <option> −y` will display a list of the available debug info formats for the given output format. The default format is indicated by an asterisk. For example:
键入 nasm-f < option >-y 将显示给定输出格式的可用调试信息格式列表。默认格式用星号表示。例如:

```
nasm −f elf −y
Nasm-f elf-y
```

```
        valid debug formats for 'elf32' output            format are
          有效的调试格式 用于" elf32"输出                  格式如下:
   ('*' denotes default):
   ('*'表示默认设定) :
        * stabs        ELF32 (i386)    stabs    debug    format    for Linux
       * stab *        ELF32 (i386)    戳       调试     格式      用于 Linux
                                                                   for
         dwarf          elf32 (i386)   dwarf    debug    format    表    Linux
          侏儒          Elf32 (i386)   侏儒     调试     格式      格    Linux
```

2.1.28 The −−`(g|l)prefix`, −−`(g|l)postfix` Options.
2.1.28-(g | l)前缀,-(g | l)后缀选项。

The `−−(g)prefix` options prepend the given argument to all `extern`, `common`, `static`, and `global` symbols, and the `−−lprefix` option prepends to all other symbols. Similarly, `−−(g)postfix` and `−−lpostfix` options append the argument in the exactly same way as the `−−xxprefix` options does.
- (g)前缀选项将给定的参数放在所有外部、公共、静态和全局符号的前面,-- lprefix 选项放在所有其他符号的前面。类似地,-(g)后缀和-lpostfix 选项以与-xxprefix 选项完全相同的方式附加参数。

Running this:
运行:

```
nasm −f macho −−gprefix _
Nasm-f macho-gprefix _
```

is equivalent to place the directive with `%pragma macho gprefix _` at the start of the file (section 6.9). It will prepend the underscore to all global and external variables, as C requires it in some, but not all, system calling conventions.
相当于在文件的开头放置带有% pragma macho gprefix _ 的指令(第 6.9 节)。它会在所有的全局变量和外部变量前面加上下划线，因为 c 在一些但不是所有的系统调用约定中要求它。

2.1.29 The −−`pragma` Option
2.1.29-杂注选项

NASM accepts an argument as `%pragma` option, which is like placing a `%pragma` preprocess statement at the beginning of the source. Running this:
NASM 接受一个参数作为% pragma 选项，这就像在源代码的开头放置一个% pragma 预处理语句。运行:

```
nasm -f macho --pragma "macho gprefix _"
Nasm-f macho-pragma〝 macho gprefix _″
```

is equivalent to the example in section 2.1.28.
等同于 2.1.28 节中的例子。

2.1.30 The `--before` Option
2.1.30 选项之前

A preprocess statement can be accepted with this option. The example shown in section 2.1.29 is the same as running this:
这个选项可以接受预处理语句。第 2.1.29 节所示的例子与运行下面的命令是一样的:

```
nasm -f macho --before "%pragma macho gprefix _"
Nasm-f macho-在〝% pragma macho gprefix _″之前
```

2.1.31 The `--limit-X` Option
2.1.31-limit-x 选项

This option allows user to setup various maximum values for these:
此选项允许用户设置各种最大值:

- `--limit-passes`: Number of maximum allowed passes. Default is effectively unlimited.

限制通行： 允许的最大通行次数。默认是有效的无限制。

- `--limit-stalled-passes`: Maximum number of allowed unfinished passes. Default is 1000.

- 限制-停止-通行证： 允许的未完成通行证的最大数量。默认值为 1000。

- `--limit-macro-levels`: Define maximum depth of macro expansion (in preprocess). Default is 1000000.

限制-宏级别： 定义宏展开的最大深度(预处理中) ，默认值为 1000000。

- `--limit-rep`: Maximum number of allowed preprocessor loop, defined under `%rep`. Default is 1000000.

`Limit-rep`: 允许的预处理器循环的最大数量，在`% rep` 下定义。 `Default` 是 `1000000`。

- `--limit-eval`: This number sets the boundary condition of allowed expression length. Default is 1000000.

`- limit-eval`: 这个数字设置允许表达式长度的边界条件。默认值为 `1000000`。

- `--limit-lines`: Total number of source lines as allowed to be processed. Default is

2000000000. In example, running this limits the maximum line count to be 1000.

限制行: 允许处理的源行总数。默认值是 `20000000000`。例如，运行此命令将最大行数限制为 `1000`。

```
nasm --limit-lines 1000
Nasm 极限线 1000
```

2.1.32 The `--keep-all` Option

2.1.32 保留所有选项

This option prevents NASM from deleting any output files even if an error happens.
此选项防止 NASM 删除任何输出文件，即使发生错误。

2.1.33 The `--no-line` Option

2.1.33 无行选项

If this option is given, all `%line` directives in the source code are ignored. This can be useful for debugging already preprocessed code. See section 4.10.1.
如果给定此选项，则忽略源代码中的所有% line 指令。这对于调试已经预处理过的代码很有用。参见第 4.10.1 节。

2.1.34 The `NASMENV` Environment Variable

2.1.34 NASMENV 环境变量

If you define an environment variable called `NASMENV`, the program will interpret it as a list of extra command−line options, which are processed before the real command line. You can use this to define standard search directories for include files, by putting `-i` options in the `NASMENV` variable.
如果您定义了一个名为 NASMENV 的环境变量，程序将把它解释为一个额外的命令行选项列表，这些选项在真正的命令行之前处理。你可以用它来定义包含文件的标准搜索目录，通过在 NASMENV 变量中放置 -i 选项。

The value of the variable is split up at white space, so that the value `-s -ic:\nasmlib\` will be treated as two separate options. However, that means that the value `-dNAME="my name"` won't do what you might want, because it will be split at the space and the NASM command−line processing will get confused by the two nonsensical words `-dNAME="my` and `name"`.
变量的值在空白处被分割，这样值 -s-ic: nasmlib 将被视为两个独立的选项。然而，这意味着值-dNAME = " my name"不会执行您可能想要的操作，因为它将在空格中被分割，而 NASM 命令行处理将会被两个无意义的单词-dNAME = " my and name"弄混。

To get round this, NASM provides a feature whereby, if you begin the `NASMENV` environment variable with some character that isn't a minus sign, then NASM will treat this character as the separator character for options. So setting the `NASMENV` variable to the value `!-s!-ic:\nasmlib\` is equivalent to setting it to `-s -ic:\nasmlib\`, but `!-dNAME="my name"` will work.
为了解决这个问题，NASM 提供了一个特性，根据这个特性，如果 NASMENV 环境变量以一个不是负号的字符开始，那么 NASM 将把这个字符作为选项的分隔符。所以把 NASMENV 变量设置为这个值！S！- ic: nasmlib 相当于将它设置为-s-ic: nasmlib，但是！dNAME = " my name"可以工作。

This environment variable was previously called `NASM`. This was changed with version 0.98.31.
这个环境变量之前被称为 NASM。这个版本被修改为 0.98.31。

2.2 Quick Start for MASM Users

2.2 MASM 用户的快速启动服务

If you’re used to writing programs with MASM, or with TASM in MASM−compatible (non−Ideal) mode, or with `a86`, this section attempts to outline the major differences between MASM’s syntax and NASM’s. If you’re not already used to MASM, it’s probably worth skipping this section.
如果您习惯于使用 MASM 编写程序，或者使用与 MASM 兼容(非理想)模式下的 TASM 编写程序，或者使用 a86 编写程序，本节将尝试概述 MASM 语法和 NASM 语法之间的主要差异。如果你还没有习惯 MASM，那么这一部分可能值得跳过。

2.2.1 NASM Is Case−Sensitive
2.2.1 NASM 区分大小写

One simple difference is that NASM is case−sensitive. It makes a difference whether you call your label `foo`, `Foo` or `FOO`. If you’re assembling to `DOS` or `OS/2 .OBJ` files, you can invoke the `UPPERCASE` directive (documented in section 7.4) to ensure that all symbols exported to other code modules are forced to be upper case; but even then, *within* a single module, NASM will distinguish between labels differing only in case.
一个简单的区别是 NASM 是区分大小写的。不管你的标签是 FOO，FOO 还是 FOO，它都是有区别的。如果你正在组装 DOS 或者 OS/2。OBJ 文件中，您可以调用大写指令(见第 7.4 节)来确保导出到其他代码模块的所有符号都必须是大写; 但即便如此，在单个模块中，NASM 也将区分只在大小写情况下不同的标签。

2.2.2 NASM Requires Square Brackets For Memory References
2.2.2 NASM 需要方括号作为内存引用

NASM was designed with simplicity of syntax in mind. One of the design goals of NASM is that it should be possible, as far as is practical, for the user to look at a single line of NASM code and tell what opcode is generated by it. You can’t do this in MASM: if you declare, for example,
NASM 设计时考虑到了语法的简单性。NASM 的设计目标之一是，在实际可行的情况下，用户可以查看一行 NASM 代码，并告诉它生成了什么操作码。你不能在 MASM 中这样做: 例如，如果你声明,

```
foo     equ
译注:   等等     1
bar     dw
酒吧    Dw       2
```

then the two lines of code
然后是两行代码

```
mov     ax,foo
Mov ax foo
mov     ax,bar
移动斧头，酒吧
```

generate completely different opcodes, despite having identical−looking syntaxes.
产生完全不同的操作码，尽管有相同的外观语法。

NASM avoids this undesirable situation by having a much simpler syntax for memory references. The rule is simply that any access to the *contents* of a memory location requires square brackets around the address, and any access to the *address* of a variable doesn't. So an instruction of the form `mov ax,foo` will *always* refer to a compile−time constant, whether it's an `EQU` or the address of a variable; and to access the *contents* of the variable `bar`, you must code `mov ax,[bar]`.
NASM 通过简化内存引用的语法避免了这种不希望出现的情况。规则很简单，对内存位置内容的任何访问都需要地址周围的方括号，而对变量地址的任何访问都不需要。所以 mov ax，foo 形式的指令总是引用编译时常量，不管是 EQU 还是变量的地址，要访问变量栏的内容，必须编码 mov ax，[bar]。

This also means that NASM has no need for MASM's `OFFSET` keyword, since the MASM code `mov ax,offset bar` means exactly the same thing as NASM's `mov ax,bar`. If you're trying to get large amounts of MASM code to assemble sensibly under NASM, you can always code `%idefine offset` to make the preprocessor treat the `OFFSET` keyword as a no−op.
这也意味着 NASM 不需要 MASM 的 OFFSET 关键字，因为 MASM 代码 mov ax，OFFSET bar 的含义与 NASM 的 mov ax，bar 完全相同。如果您试图获得大量 MASM 代码以便在 NASM 下合理地汇编，那么您总是可以编码% idedefine OFFSET 以使预处理器将 OFFSET 关键字视为 no-op。

This issue is even more confusing in `a86`, where declaring a label with a trailing colon defines it to be a 'label' as opposed to a 'variable' and causes `a86` to adopt NASM−style semantics; so in `a86`, `mov ax,var` has different behaviour depending on whether `var` was declared as `var: dw 0` (a label) or `var dw 0` (a word−size variable). NASM is very simple by comparison: *everything* is a label.
在 a86 中，这个问题更加令人困惑，在 a86 中，用后缀冒号声明一个标签将其定义为' label'，而不是' variable'，并导致 a86 采用 NASM 风格的语义; 因此，在 a86 中，mov ax，var 具有不同的行为，这取决于是将 var 声明为 var: dw 0(一个标签)还是 var dw 0(一个字大小变量)。相比之下，NASM 非常简单: 所有东西都是一个标签。

NASM, in the interests of simplicity, also does not support the hybrid syntaxes supported by MASM and its clones, such as `mov ax,table[bx]`, where a memory reference is denoted by one portion outside square brackets and another portion inside. The correct syntax for the above is `mov ax,[table+bx]`. Likewise, `mov ax,es:[di]` is wrong and `mov ax,[es:di]` is right.
为了简单起见，NASM 也不支持 MASM 及其克隆支持的混合语法，如 mov ax，table [bx]，其中内存引用由方括号外的一部分和方括号内的另一部分表示。上面的正确语法是 mov ax，[table + bx]。同样，mov ax，es: [di]是错误的，mov ax，[es: di]是正确的。

2.2.3 NASM Doesn't Store Variable Types
2.2.3 NASM 不存储变量类型

NASM, by design, chooses not to remember the types of variables you declare. Whereas MASM will remember, on seeing `var dw 0`, that you declared `var` as a word−size variable, and will then be able to fill in the ambiguity in the size of the instruction `mov var,2`, NASM will deliberately remember nothing about the symbol `var` except where it begins, and so you must explicitly code `mov word [var],2`.
NASM 通过设计选择不记住您声明的变量类型。MASM 会记住，在看到 var dw 0 时，你将 var 声明为一个单词大小变量，然后能够填充指令 mov var，2 大小的模糊性，而 NASM 将故意不记住任何关于符号 var 的内容，除非它从哪里开始，所以你必须显式地编码 mov word [var]，2。

For this reason, NASM doesn't support the `LODS`, `MOVS`, `STOS`, `SCAS`, `CMPS`, `INS`, or `OUTS` instructions, but only supports the forms such as `LODSB`, `MOVSW`, and `SCASD`, which explicitly specify the size of the components of the strings being manipulated.
出于这个原因，NASM 不支持 LODS、 MOVS、 STOS、 SCAS、 CMPS、 INS 或 OUTS 指令，而只支持 LODSB、 MOVSW 和 SCASD 等形式，这些形式显式地指定了被操作字符串组件的大小。

2.2.4 NASM Doesn't `ASSUME`
2.2.4 NASM 不假设

As part of NASM's drive for simplicity, it also does not support the `ASSUME` directive. NASM will not keep track of what values you choose to put in your segment registers, and will never *automatically* generate a segment override prefix.
作为 NASM 追求简单性的一部分，它也不支持 ASSUME 指令。NASM 不会跟踪你选择放入段寄存器的值，也不会自动生成段覆盖前缀。

2.2.5 NASM Doesn't Support Memory Models
2.2.5 NASM 不支持内存模型

NASM also does not have any directives to support different 16−bit memory models. The programmer has to keep track of which functions are supposed to be called with a far call and which with a near call, and is responsible for putting the correct form of `RET` instruction (`RETN` or `RETF`; NASM accepts `RET` itself as an alternate form for `RETN`); in addition, the programmer is responsible for coding CALL FAR instructions where necessary when calling *external* functions, and must also keep track of which external variable definitions are far and which are near.
NASM 也没有任何支持不同 16 位内存模型的指令。程序员必须跟踪哪些函数应该用远调用调用，哪些函数应该用近调用调用，并且负责放置 RET 指令的正确形式(RETN 或 RETF; NASM 接受 RET 本身作为 RETN 的替代形式) ; 此外，程序员在调用外部函数时，必要时负责编写 CALL FAR 指令，并且还必须跟踪哪些外部变量定义是远的，哪些是近的。

2.2.6 Floating−Point Differences
2.2.6 浮点差异

NASM uses different names to refer to floating−point registers from MASM: where MASM would call them `ST(0)`, `ST(1)` and so on, and `a86` would call them simply `0`, `1` and so on, NASM chooses to call them `st0`, `st1` etc.
NASM 使用不同的名称来指代来自 MASM 的浮点寄存器: MASM 将其称为 ST (0)、 ST (1)等，a86 将其称为简单的 0、1 等，NASM 选择将其称为 st0、 st1 等。

As of version 0.96, NASM now treats the instructions with 'nowait' forms in the same way as MASM−compatible assemblers. The idiosyncratic treatment employed by 0.95 and earlier was based on a misunderstanding by the authors.
从 0.96 版本开始，NASM 现在以与 MASM 兼容的汇编程序相同的方式处理带有" nowait"表单的指令。0.95 及更早版本使用的特殊处理方法是基于作者的误解。

2.2.7 Other Differences
2.2.7 其他差异

For historical reasons, NASM uses the keyword `TWORD` where MASM and compatible assemblers use `TBYTE`.
由于历史原因，NASM 使用关键字 TWORD，而 MASM 和兼容汇编器使用 TBYTE。

NASM does not declare uninitialized storage in the same way as MASM: where a MASM programmer might use `stack db 64 dup (?)`, NASM requires `stack resb 64`, intended to be read as 'reserve 64 bytes'. For a limited amount of compatibility, since NASM treats `?` as a valid character in symbol names, you can code `? equ 0` and then writing `dw ?` will at least do something vaguely useful. `DUP` is still not a supported syntax, however.
NASM 不像 MASM 那样声明未初始化的存储: MASM 程序员可能使用堆栈 db64dup (?)NASM 需要栈 resb 64，意思是"预留 64 字节"。对于有限的兼容性，因为 NASM 处理？作为一个有效的字符在符号名称，你可以编码？Equ 0 然后写 dw？至少会做一些有用的东西。然而，DUP 仍然不是一个受支持的语法。

In addition to all of this, macros and directives work completely differently to MASM. See chapter 4 and chapter 6 for further details.
除此之外，宏和指令的工作方式与 MASM 完全不同。详情请参阅第 4 章和第 6 章。

29

30

Chapter 3: The NASM Language
第三章: NASM 语言

3.1 Layout of a NASM Source Line
3.1 NASM 源代码行的布局

Like most assemblers, each NASM source line contains (unless it is a macro, a preprocessor directive or an assembler directive: see chapter 4 and chapter 6) some combination of the four fields
像大多数汇编程序一样，每个 NASM 源代码行包含(除非是宏、预处理程序指令或汇编程序指令: 参见第 4 章和第 6 章)四个字段的一些组合

```
label:    instruction operands        ; comment
Label: 指令操作数; 注释
```

As usual, most of these fields are optional; the presence or absence of any combination of a label, an instruction and a comment is allowed. Of course, the operand field is either required or forbidden by the presence and nature of the instruction field.
像往常一样，这些字段中的大多数都是可选的; 允许存在或不存在标签、指令和注释的任何组合。当然，由于指令字段的存在和性质，操作数字段是必需的或禁止的。

NASM uses backslash (\) as the line continuation character; if a line ends with backslash, the next line is considered to be a part of the backslash−ended line.
NASM 使用反斜杠()作为行的延续字符; 如果一行以反斜杠结束，则下一行被认为是反斜杠结束行的一部分。

NASM places no restrictions on white space within a line: labels may have white space before them, or instructions may have no space before them, or anything. The colon after a label is also optional. (Note that this means that if you intend to code `lodsb` alone on a line, and type `lodab` by accident, then that's still a valid source line which does nothing but define a label. Running NASM with the command−line option `-w+orphan-labels` will cause it to warn you if you define a label alone on a line without a trailing colon.)
NASM 对行内的空白没有限制: 标签前面可能有空白，或者说明前面可能没有空白，或者其他任何东西。标签后面的冒号也是可选的。(请注意，这意味着如果您打算在一行上单独编码 lodsb，并偶然键入 lodab，那么这仍然是一个有效的源代码行，它只定义一个标签。使用命令行选项 -w + 孤儿标签运行 NASM 时，如果您在一行中单独定义一个标签，而后面没有冒号，则会导致 NASM 发出警告。)

Valid characters in labels are letters, numbers, `_`, `$`, `#`, `@`, `~`, `.`, and `?`. The only characters which may be used as the *first* character of an identifier are letters, `.` (with special meaning: see section 3.9), `_` and `?`. An identifier may also be prefixed with a `$` to indicate that it is intended to be read as an identifier and not a reserved word; thus, if some other module you are linking with defines a symbol called `eax`, you can refer to `$eax` in NASM code to distinguish the symbol from the register. Maximum length of an identifier is 4095 characters.
标签中有效的字符是字母，数字，_，$，# ,@，~ 。和？.唯一可以用作标识符第一个字符的字符是字母。(特殊含义: 参见第 3.9 节)，_ 和？.标识符也可以加上 $的前缀，以表明该标识符是作为标识符而不是保留字读取的; 因此，如果与您链接的其他模块定义了一个称为 eax 的符号，则可以在 NASM 代码中引用 $eax 来区分该符号和寄存器。一个标识符的最大长度是 4095 个字符。

The instruction field may contain any machine instruction: Pentium and P6 instructions, FPU instructions, MMX instructions and even undocumented instructions are all supported. The instruction may be prefixed by `LOCK`, `REP`, `REPE`/`REPZ`, `REPNE`/`REPNZ`, `XACQUIRE`/`XRELEASE` or `BND`/`NOBND`, in the usual way. Explicit address−size and operand−size prefixes `A16`, `A32`, `A64`, `O16` and `O32`, `O64` are provided – one example of their use is given in chapter 10. You can also use the name of a segment register as an instruction prefix: coding `es mov [bx],ax` is equivalent to coding `mov [es:bx],ax`. We recommend the latter syntax, since it is consistent with other syntactic features of the language, but for instructions such as `LODSB`, which has no operands and yet can require a segment override, there is no clean syntactic way to proceed apart from `es lodsb`.

指令字段可以包含任何机器指令: Pentium 和 p6 指令，FPU 指令，MMX 指令，甚至非文档化的指令都支持。该指令可以以通常的方式以 LOCK、 REP、 REPE/REPZ、 REPNE/REPNZ、XACQUIRE/XRELEASE 或 BND/NOBND 作为前缀。提供了显式的地址大小和操作数大小前缀 A16、A32、 A64、 o16 和 O32、 O64——第 10 章给出了它们的一个使用示例。您还可以使用段寄存器的名称作为指令前缀: 编码 es mov [bx]，ax 等效于编码 mov [es: bx]，ax。我们推荐后一种语法，因为它与语言的其他语法特性是一致的，但是对于像 LODSB 这样没有操作数但是需要重写段的指令来说，除了 es LODSB 之外没有其他简洁的语法方式。

An instruction is not required to use a prefix: prefixes such as `CS`, `A32`, `LOCK` or `REPE` can appear on a line by themselves, and NASM will just generate the prefix bytes.
指令不需要使用前缀: 诸如 CS、 A32、 LOCK 或 REPE 之类的前缀可以单独出现在一行中，而 NASM 只生成前缀字节。

In addition to actual machine instructions, NASM also supports a number of pseudo−instructions, described in section 3.2.
除了实际的机器指令之外，NASM 还支持 3.2 节中描述的许多伪指令。

Instruction operands may take a number of forms: they can be registers, described simply by the register name (e.g. `ax`, `bp`, `ebx`, `cr0`: NASM does not use the `gas`–style syntax in which register names must be prefixed by a `%` sign), or they can be effective addresses (see section 3.3), constants (section 3.4) or expressions (section 3.5).
指令操作数可以采用多种形式: 它们可以是寄存器，只用寄存器名称来描述(例如 ax、 bp、 ebx、cr0: NASM 不使用必须以% 符号作为前缀的气体式语法)，也可以是有效地址(参见 3.3 节)、常量(参见 3.4 节)或表达式(参见 3.5 节)。

For x87 floating−point instructions, NASM accepts a wide range of syntaxes: you can use two−operand forms like MASM supports, or you can use NASM's native single−operand forms in most cases. For example, you can code:
对于 x87 浮点指令，NASM 接受广泛的语法: 您可以使用像 MASM 支持的两个操作数形式，或者在大多数情况下可以使用 NASM 的原生单个操作数形式。例如，你可以编码:

```
fadd    st1             ; this sets st0 := st0 + st1
Fadd st1; 设置 st0: = st0 + st1
fadd    st0,st1         ; so does this
Fadd st0, st1; 这样也可以
```

```
fadd    st1,st0          ; this sets st1 := st1 + st0
Fadd st1, st0; 这将设置 st1: = st1 + st0
fadd    to st1           ; so does this
Fadd 到 st1; 这样做
```

Almost any x87 floating−point instruction that references memory must use one of the prefixes `DWORD`, `QWORD` or `TWORD` to indicate what size of memory operand it refers to.
几乎任何引用内存的 x87 浮点指令都必须使用前缀 DWORD、 QWORD 或 TWORD 之一来指示引用的内存操作数的大小。

3.2 Pseudo−Instructions
3.2 伪指令

Pseudo−instructions are things which, though not real x86 machine instructions, are used in the instruction field anyway because that's the most convenient place to put them. The current pseudo−instructions are `DB`, `DW`, `DD`, `DQ`, `DT`, `DO`, `DY` and `DZ`; their uninitialized counterparts `RESB`, `RESW`, `RESD`, `RESQ`, `REST`, `RESO`, `RESY` and `RESZ`; the `INCBIN` command, the `EQU` command, and the `TIMES` prefix.
虽然不是真正的 x86 机器指令，但是伪指令是指令字段中使用的东西，因为那是放置它们最方便的地方。当前的伪指令是 DB、 DW、 DD、 DQ、 DT、 DO、 DY 和 DZ; 它们未初始化的对应指令是 RESB、 RESW、 RESD、 RESQ、 REST、 resso、 RESY 和 RESZ; INCBIN 命令、 EQU 命令和 TIMES 前缀。

3.2.1 `DB` and Friends: Declaring Initialized Data
3.2.1 DB 和 Friends: 声明已初始化的数据

`DB`, `DW`, `DD`, `DQ`, `DT`, `DO`, `DY` and `DZ` are used, much as in MASM, to declare initialized data in the output file. They can be invoked in a wide range of ways:
使用 DB、 DW、 DD、 DQ、 DT、 DO、 DY 和 DZ 来声明输出文件中的初始化数据，这与 MASM 中的情况非常相似。它们可以以多种方式被调用:

```
db
数据   0x55                ; just the byte 0x55
库     0x55                只有字节 0x55
db
数据   0x55,0x56,0x57      ; three bytes in succession
库     0x55,0 x56,0 x57    连续三个字节
db
数据   'a',0x55            ; character      constants are OK
库     A', 0 x55           ; 角色           常数没问题
db
数据   'hello',13,10,'$'   ; so are string constants
库     '你好', 13,10,'$    字符串常量也是
                               ; 0x34 0x12
dw     0x1234              第三季, 第 12
Dw     0x1234                        集
                           ; 0x61    0x00
                           第六季      第
dw     'a'                 第六十    0x00 (it's just a number)
Dw     " a"                一集        集 (这只是一个数字)
                           ; 0x61    0x62
                           第六季      第
dw     'ab'                第六十    0x62 (character constant)
Dw     " ab"               一集        集 (字符常数)
```

		; 0x61	0x62
		第六季	第
dw	’abc’	第六十	0x62 0x63 0x00 (string)
Dw	“ abc”	一集	集 0x630x00(字符串)
dd			
编		; 0x78	
译:		第七	
pest	0x12345678	季，第	0x56 0x34 0x12
wave	0x12345678	78 集	0x56 第三季，第 12 集
dd			
编			
译:			
pest	1.234567e20	; floating−point constant	
wave	第一季，第 20 集	; 浮点常数	
dq	0x123456789abcdef0	; eight byte constant	
Dq	0x123456789abcdef0	; 八字节常量	
dq	1.234567e20	; double−precision float	
Dq	第一季，第 20 集	; 双精度浮点数	
dt	1.234567e20	; extended−precision float	
Dt	第一季，第 20 集	; 扩展精度浮子	

`DT`, `DO`, `DY` and `DZ` do not accept numeric constants as operands.
`DT`，`DO`，`DY` 和 `DZ` 不接受数值常量作为操作数。

3.2.2 `RESB` and Friends: Declaring Uninitialized Data
3.2.2 RESB 和 Friends: 声明未初始化的数据

`RESB`, `RESW`, `RESD`, `RESQ`, `REST`, `RESO`, `RESY` and `RESZ` are designed to be used in the BSS section of a module: they declare *uninitialized* storage space. Each takes a single operand, which is the number of bytes, words, doublewords or whatever to reserve. As stated in section 2.2.7, NASM does not support the MASM/TASM syntax of reserving uninitialized space by writing `DW ?` or similar things: this is what it does instead. The operand to a `RESB`–type pseudo−instruction is a *critical expression*: see section 3.8.
`RESB`、 `RESW`、 `RESD`、 `RESQ`、 `REST`、 `RESO`、 `RESY` 和 `RESZ` 被设计用于模块的 `BSS` 部分：它们声明未初始化的存储空间。每个操作数只有一个操作数，即字节数、单词数、双词数或任何需要保留的数目。如第 `2.2.7` 节所述，`NASM` 不支持通过写入 `DW` 来保留未初始化空间的 `MASM/TASM` 语法？或者类似的东西：这就是它所做的。`RESB` 类型伪指令的操作数是一个关键表达式：参见第 `3.8` 节。

For example:
例如:

buffer:	resb		; reserve 64 bytes
缓冲器:	Resb	64	; 预留 64 字节
wordvar:	resw		; reserve a word
译者:	翻译	1	保留一句话
realarray	resq		; array of ten reals
真实数组	Resq	10	; 十个真实的数组
ymmval:	resy		; one YMM register
Ymmval:	雷西	1	; 一个 YMM 寄存器
zmmvals:	resz		; 32 ZMM registers
Zmmvals:	再见	32	; 32 个 ZMM 寄存器

3.2.3 `INCBIN`: Including External Binary Files
3.2.3 INCBIN: 包括外部二进制文件

`INCBIN` is borrowed from the old Amiga assembler DevPac: it includes a binary file verbatim into the output file. This can be handy for (for example) including graphics and sound data directly into a game executable file. It can be called in one of these three ways:
INCBIN 是从老的 Amiga 汇编程序 DevPac 借来的：它在输出文件中包含了一个二进制文件。这对于(例如)将图形和声音数据直接包含到游戏可执行文件中是很方便的。它可以通过以下三种方式之一来调用：

```
    incbin  "file.dat"             ; include the whole file
    Incbin“ file.dat”; 包括整个文件
    incbin  "file.dat",1024        ; skip the first 1024 bytes
    Incbin“ file.dat”，1024; 跳过前面的 1024 字节
```

32

```
incbin  "file.dat",1024,512                skip the      first 1024, and
Incbin“ file.dat”， 1024,512             ; 跳过          第 1024 项，以及
                                           actually      include at most
                                         ; 事实上        最多包括          512
```

INCBIN is both a directive and a standard macro; the standard macro version searches for the file in the include file search path and adds the file to the dependency lists. This macro can be overridden if desired.
INCBIN 既是一个指令，也是一个标准宏；标准宏版本在包含文件搜索路径中搜索文件，并将文件添加到依赖项列表中。如果需要，这个宏可以被重写。

3.2.4 EQU: Defining Constants
3.2.4 EQU: 定义常量

EQU defines a symbol to a given constant value: when EQU is used, the source line must contain a label. The action of EQU is to define the given label name to the value of its (only) operand. This definition is absolute, and cannot change later. So, for example,
EQU 将符号定义为给定的常量值：当使用 EQU 时，源代码行必须包含一个标签。EQU 的作用是将给定的标签名称定义为它的(唯一的)操作数的值。这个定义是绝对的，以后不能更改。举个例子,

```
message         db      'hello, world'
消息 db‘ hello，world'
msglen          equ     $-message
Msglen equ 消息
```

defines msglen to be the constant 12. msglen may not then be redefined later. This is not a preprocessor definition either: the value of msglen is evaluated *once*, using the value of $ (see section 3.5 for an explanation of $) at the point of definition, rather than being evaluated wherever it is referenced and using the value of $ at the point of reference.
定义 msglen 为常数 12。Msglen 以后可能不会被重新定义。这也不是一个预处理器定义: msglen 的值只计算一次，在定义点使用 $的值(参见第 3.5 节对 $的解释)，而不是在引用它的地方计算，在引用点使用 $的值。

3.2.5 TIMES: Repeating Instructions or Data
3.2.5 次: 重复指令或数据

The TIMES prefix causes the instruction to be assembled multiple times. This is partly present as NASM's equivalent of the DUP syntax supported by MASM–compatible assemblers, in that you can code
TIMES 前缀导致指令被多次组装。这在一定程度上表现为 NASM 支持的与 MASM 兼容的汇编程序所支持的 DUP 语法的等价物，因为你可以编码

```
zerobuf:        times 64 db 0
Zerobuf: 乘以 64 分贝 0
```

or similar things; but TIMES is more versatile than that. The argument to TIMES is not just a numeric constant, but a numeric *expression*, so you can do things like
或者类似的东西; 但是 TIMES 的功能要比这个多得多。TIMES 的参数不仅仅是一个数值常量，而是一个数值表达式，所以你可以这样做

```
buffer: db      'hello, world'
缓冲区: db‘ hello，world'
        times 64-$+buffer db ' '
        乘以 64-$+ buffer db'
```

which will store exactly enough spaces to make the total length of buffer up to 64. Finally, TIMES can be applied to ordinary instructions, so you can code trivial unrolled loops in it:
它将存储足够的空间，使缓冲区的总长度达到 64。最后，TIMES 可以应用于普通的指令，所以你可以在其中编写无关紧要的展开循环代码:

```
        times 100 movsb
        乘以 100 个 movsb
```

Note that there is no effective difference between `times 100 resb 1` and `resb 100`, except that the latter will be assembled about 100 times faster due to the internal structure of the assembler.
请注意，乘以 100 resb 1 和 resb 100 之间没有有效的区别，除了后者由于汇编程序的内部结构，组装速度将快 100 倍左右。

The operand to `TIMES` is a critical expression (section 3.8).
TIMES 的操作数是一个关键表达式(第 3.8 节)。

Note also that `TIMES` can't be applied to macros: the reason for this is that `TIMES` is processed after the macro phase, which allows the argument to `TIMES` to contain expressions such as `64-$+buffer` as above. To repeat more than one line of code, or a complex macro, use the preprocessor `%rep` directive.
还要注意，TIMES 不能应用于宏: 原因是 TIMES 是在宏阶段之后处理的，这允许 TIMES 的参数包含如上所述的 64-$+ buffer 之类的表达式。要重复多行代码，或者一个复杂的宏，使用预处理器% rep 指令。

3.3 Effective Addresses
3.3 有效地址

An effective address is any operand to an instruction which references memory. Effective addresses, in NASM, have a very simple syntax: they consist of an expression evaluating to the desired address, enclosed in square brackets. For example:
有效地址是指向引用内存的指令的任何操作数。在 NASM 中，有效地址有一个非常简单的语法: 它们包含一个计算所需地址的表达式，括在方括号中。例如:

```
wordvar dw      123
Wordvar dw 123
        mov     ax,[wordvar]
        Mov ax, [ wordvar ]
        mov     ax,[wordvar+1]
        Mov ax, [ wordvar + 1]
        mov     ax,[es:wordvar+bx]
        Mov ax, [ es: wordvar + bx ]
```

Anything not conforming to this simple system is not a valid memory reference in NASM, for example `es:wordvar[bx]`.
任何不符合这个简单系统的东西在 NASM 中都不是有效的内存引用，例如 es: wordvar [bx]。

More complicated effective addresses, such as those involving more than one register, work in exactly the same way:
更复杂的有效地址，例如那些涉及多个寄存器的地址，工作原理完全相同:

```
mov     eax,[ebx*2+ecx+offset]
[ ebx * 2 + ecx + 偏移]
mov     ax,[bp+di+8]
Mov ax [ bp + di + 8]
```

NASM is capable of doing algebra on these effective addresses, so that things which don't necessarily *look* legal are perfectly all right:
NASM 能够对这些有效地址进行代数运算，所以看起来不一定合法的东西完全没问题:

```
mov     eax,[ebx*5]             ; assembles as [ebx*4+ebx]
Mov eax, [ ebx * 5] ; 汇编为[ ebx * 4 + ebx ]
mov     eax,[label1*2-label2]   ; ie [label1+(label1-label2)]
Mov eax, [ label1 * 2-label2] ; 即[ label1 + (label1-label2)]
```

Some forms of effective address have more than one assembled form; in most such cases NASM will generate the smallest form it can. For example, there are distinct assembled forms for the 32−bit effective addresses `[eax*2+0]` and `[eax+eax]`, and NASM will generally generate the latter on the grounds that the former requires four bytes to store a zero offset.
一些有效地址的形式有不止一种组合形式; 在大多数情况下，NASM 会生成它能生成的最小形式。例如，32 位有效地址[eax * 2 + 0]和[eax + eax]有不同的组合形式，NASM 通常会生成后者，理由是前者需要 4 个字节来存储零偏移量。

NASM has a hinting mechanism which will cause `[eax+ebx]` and `[ebx+eax]` to generate different opcodes; this is occasionally useful because `[esi+ebp]` and `[ebp+esi]` have different default segment registers.
NASM 有一个提示机制，它会导致[eax + ebx]和[ebx + eax]生成不同的操作码; 这有时是有用的，因为[esi + ebp]和[ebp + esi]有不同的默认段寄存器。

However, you can force NASM to generate an effective address in a particular form by the use of the keywords `BYTE`, `WORD`, `DWORD` and `NOSPLIT`. If you need `[eax+3]` to be assembled using a double−word offset field instead of the one byte NASM will normally generate, you can code `[dword eax+3]`. Similarly, you can force NASM to use a byte offset for a small value which it hasn't seen on the first pass (see section 3.8 for an example of such a code fragment) by using `[byte eax+offset]`. As special cases, `[byte eax]` will code `[eax+0]` with a byte offset of zero, and `[dword eax]` will code it with a double−word offset of zero. The normal form, `[eax]`, will be coded with no offset field.
然而，你可以通过使用 BYTE，WORD，DWORD 和 NOSPLIT 这些关键字强制 NASM 生成一个特定形式的有效地址。如果您需要[eax + 3]使用双字偏移量字段而不是 NASM 通常生成的一个字节进行组装，您可以编写[dword eax + 3]。类似地，您可以通过使用[byte eax + offset]来强制 NASM 为第一次通过时没有看到的小值使用字节偏移量(参见第 3.8 节中这样的代码片段的示例)。作为特殊情况，[byte eax]将使用零的字节偏移量来编码[eax + 0]，而[dword eax]将使用零的双字偏移量来编码它。正常形式[eax]将被编码为没有偏移量字段。

The form described in the previous paragraph is also useful if you are trying to access data in a 32−bit segment from within 16 bit code. For more information on this see the section on mixed−size addressing (section 10.2). In particular, if you need to access data with a known offset that is larger than will fit in a 16−bit value, if you don't specify that it is a dword offset, nasm will cause the high word of the offset to be lost.
如果您试图从 16 位代码中访问 32 位段中的数据，那么上一段描述的表单也很有用。更多信息请参见混合大小寻址部分(第 10.2 节)。特别是，如果您需要访问具有大于 16 位值的已知偏移量的数据，如果您不指定它是 dword 偏移量，则 nasm 将导致偏移量的高字丢失。

Similarly, NASM will split `[eax*2]` into `[eax+eax]` because that allows the offset field to be absent and space to be saved; in fact, it will also split `[eax*2+offset]` into `[eax+eax+offset]`. You can combat this behaviour by the use of the `NOSPLIT` keyword: `[nosplit eax*2]` will force `[eax*2+0]` to be generated literally. `[nosplit eax*1]` also has the same effect. In another way, a split EA

form [0, eax*2] can be used, too. However, NOSPLIT in [nosplit eax+eax] will be ignored because user's intention here is considered as [eax+eax].
同样，NASM 将[eax * 2]分割为[eax + eax]，因为这样可以不存在偏移量字段并节省空间; 实际上，它也将[eax * 2 + offset]分割为[eax + eax + offset]。您可以通过使用 NOSPLIT 关键字来对抗这种行为: [NOSPLIT eax * 2]将强制按字面生成[eax * 2 + 0]。[nosplit eax * 1]也具有相同的效果。另一方面，分割 EA 表单[0，eax * 2]也可以使用。然而，NOSPLIT 在[NOSPLIT eax + eax]中将被忽略，因为用户的意图在这里被认为是[eax + eax]。

In 64−bit mode, NASM will by default generate absolute addresses. The REL keyword makes it produce RIP–relative addresses. Since this is frequently the normally desired behaviour, see the DEFAULT directive (section 6.2). The keyword ABS overrides REL.
在 64 位模式下，NASM 将默认生成绝对地址。REL 关键字使它产生 RIP 相对地址。由于这通常是正常需要的行为，请参阅 DEFAULT 指令(第 6.2 节)。关键字 ABS 覆盖 REL。

A new form of split effective addres syntax is also supported. This is mainly intended for mib operands as used by MPX instructions, but can be used for any memory reference. The basic concept of this form is splitting base and index.
还支持一种新的有效分割地址语法形式。这主要用于 MPX 指令所使用的 mib 操作数，但也可用于任何内存引用。这种形式的基本概念是拆分基础和索引。

```
     mov eax,[ebx+8,ecx*4]   ; ebx=base, ecx=index, 4=scale, 8=disp
   [ ebx + 8，ecx * 4] ; ebx = base，ecx = index，4 = scale，8 = disp
```

For mib operands, there are several ways of writing effective address depending on the tools. NASM supports all currently possible ways of mib syntax:
对于 mib 操作数，根据工具，有几种写有效地址的方法。NASM 支持目前所有可能的 mib 语法:

```
    ; bndstx
    Bndstx
    ; next 5 lines are parsed same
    接下来的 5 行解析结果相同
    ; base=rax, index=rbx, scale=1, displacement=3
    Base = rax，index = rbx，scale = 1，displacement = 3
    bndstx [rax+0x3,rbx], bnd0      ; NASM - split EA
    Bndstx [ rax + 0 x3，rbx ] ，bnd0; NASM 分割 EA
```

```
bndstx [rbx*1+rax+0x3], bnd0      ; GAS - '*1' indecates an index reg
bndstx [rax+rbx+3], bnd0          GAS-'* 1'表示一个索引条例
bndstx [rax+0x3], bnd0, rbx       ; GAS - without hints
bndstx [rax+0x3], rbx, bnd0       GAS-没有提示
Bndstx [ rbx * 1 + rax + 0        ; ICC-1
x3] ，bnd0 bndstx [ rax + rbx     ICC-1
+ 3] ，bnd0 bndstx [ rax + 0      ; ICC-2
x3] ，bnd0，rbx bndstx [ rax +    ICC-2
0 x3] ，rbx，bnd0
```

When broadcasting decorator is used, the opsize keyword should match the size of each element.
当使用广播修饰符时，opsize 关键字应该匹配每个元素的大小。

```
VDIVPS zmm4, zmm5, dword [rbx]{1to16}   ; single-precision float
VDIVPS zmm4，zmm5，dword [ rbx ]{1to16} ; 单精度浮点数
VDIVPS zmm4, zmm5, zword [rbx]          ; packed 512 bit memory
VDIVPS zmm4，zmm5，zword [ rbx ] ; 封装 512 位内存
```

3.4 Constants
3.4 常量

NASM understands four different types of constant: numeric, character, string and floating−point.
NASM 理解四种不同类型的常量: 数字、字符、字符串和浮点。

3.4.1 Numeric Constants
3.4.1 数值常量

A numeric constant is simply a number. NASM allows you to specify numbers in a variety of number bases, in a variety of ways: you can suffix `H` or `X`, `D` or `T`, `Q` or `O`, and `B` or `Y` for hexadecimal, decimal, octal and binary respectively, or you can prefix `0x`, for hexadecimal in the style of C, or you can prefix `$` for hexadecimal in the style of Borland Pascal or Motorola Assemblers. Note, though, that the `$` prefix does double duty as a prefix on identifiers (see section 3.1), so a hex number prefixed with a `$` sign must have a digit after the `$` rather than a letter. In addition, current versions of NASM accept the prefix `0h` for hexadecimal, `0d` or `0t` for decimal, `0o` or `0q` for octal, and `0b` or `0y` for binary. Please note that unlike C, a `0` prefix by itself does *not* imply an octal constant!
数字常量只是一个数字。NASM 允许您以各种方式以各种数字基数指定数字: 您可以将 h 或 x、 d 或 t、q 或 o 和 b 或 y 分别作为十六进制、十进制、八进制和二进制的后缀，或者您可以将 0x 作为十六进制的前缀，或者您可以将 $作为十六进制的前缀，作为 Borland Pascal 或 Motorola assembler 风格的前缀。但请注意，$前缀作为标识符的前缀具有双重作用(参见第 3.1 节)，因此以 $符号为前缀的十六进制数必须在 $后面有一个数字，而不是一个字母。此外，当前版本的 NASM 接受前缀 0h 表示十六进制，0d 或 0t 表示十进制，0o 或 0q 表示八进制，0b 或 0y 表示二进制。请注意，与 c 不同，0 前缀本身并不意味着一个八进制常量!

Numeric constants can have underscores (_) interspersed to break up long strings.
Numeric 常量可以用下划线(_)来分隔长的字符串。

Some examples (all producing exactly the same code):
一些例子(都产生完全相同的代码) :

```
mov      ax,200          ; decimal
动起来   斧头 200        十进制
mov      ax,0200         ; still decimal
动起来   Ax，0200        仍然是小数
mov      ax,0200d        ; explicitly decimal
动起来   斧头，0200 天   ; 显式十进制
mov      ax,0d200        ; also decimal
动起来   Ax，0 d200      ; 也是小数
mov      ax,0c8h         ; hex
动起来   Ax，0 c8h       十六进制
```

```
mov      ax,$0c8            ; hex again: the 0 is required
动起来   Ax，$0c8            还是十六进制: 需要 0
mov      ax,0xc8            ; hex yet again
动起来   Ax，0 xc8           再次使用巫术
mov      ax,0hc8            ; still hex
动起来   Ax，0 hc8           仍然是巫术
mov      ax,310q            ; octal
动起来   Ax，310 q           八进制
mov      ax,310o            ; octal again
动起来   斧头 310 度          又是八进制
mov      ax,0o310           ; octal yet again
动起来   Ax，0 o310          又是八进制
mov      ax,0q310           ; octal yet again
动起来   Ax，0 q310          又是八进制
mov      ax,11001000b       ; binary
动起来   Ax，11001000 b      二进制
mov      ax,1100_1000b      ; same binary constant
动起来   Ax，1100 _ 1000b    相同的二进制常数
         ax,1100_1000y
mov      斧头，1100 _ 1000   ; same binary constant once more
动起来   年                  同样的二进制常数
mov      ax,0b1100_1000     ; same binary constant yet again
动起来   Ax，0b1100 _ 1000   同样的二进制常数
mov      ax,0y1100_1000     ; same binary constant yet again
动起来   Ax，0y1100 _ 1000   同样的二进制常数
```

3.4.2 Character Strings
3.4.2 字符串

A character string consists of up to eight characters enclosed in either single quotes (`'...'`), double quotes (`"..."`) or backquotes (`` `...` ``). Single or double quotes are equivalent to NASM (except of course that surrounding the constant with single quotes allows double quotes to appear within it and vice versa); the contents of those are represented verbatim. Strings enclosed in backquotes support C−style \–escapes for special characters.
一个字符串最多由 8 个字符组成，包括单引号(“ ...”)、双引号(“ ...”)或回引号(“ ...”)。单引号或双引号等效于 NASM (当然，使用单引号围绕常量允许双引号出现在其中，反之亦然) ; 这些引号的内容是逐字表示的。用后引号括起来的字符串支持 c 风格的特殊字符转义。

The following escape sequences are recognized by backquoted strings:
下列转义序列由后引号字符串识别:

```
\'           single quote (')
单引号(')
\"           double quote (")
双引号(")
```

35

\\`	backquote (`)	
原文引用		
\\\\backslash (\\)		
反斜杠()		
\\?	question mark	(?)
\\?	问号	(?)
\\a	BEL (ASCII 7)	
A	BEL (ASCII 7)	
	BS	
\\b	胡 (ASCII 8)	
B	扯 (ASCII 8)	
	TAB	
\\t	标 (ASCII 9)	
T	签 (ASCII 9)	
\\n	LF (ASCII 10)	
N	LF (ASCII 10)	
\\v	VT (ASCII 11)	
V	VT (ASCII 11)	
	FF	
	法	
\\f	新 (ASCII 12)	
F	社 (ASCII 12)	
\\r	CR (ASCII 13)	
R	CR (ASCII 13)	
\\e	ESC (ASCII 27)	
E	ESC (ASCII 27)	
\\377	Up to 3 octal	digits − literal byte
377	高达 3 个八进制	数字-字面字节
\\xFF	Up to 2 hexadecimal digits − literal byte	
xFF	最多 2 个十六进制数字-字面字节	
\\u1234	4 hexadecimal	digits − Unicode character
U1234	4 十六进制	数字-unicode 字符
\\U12345678	8 hexadecimal	digits − Unicode character
U12345678	8 十六进制	数字-unicode 字符

All other escape sequences are reserved. Note that `\0`, meaning a `NUL` character (ASCII 0), is a special case of the octal escape sequence.
保留所有其他转义序列。注意，0，表示一个 NUL 字符(ASCII 0)，是八进制转义序列的一种特殊情况。

Unicode characters specified with `\u` or `\U` are converted to UTF−8. For example, the following lines are all equivalent:
用 u 或 u 指定的 Unicode 字符被转换为 UTF-8。例如，下面的行都是等价的:

```
db '\u263a'                  ; UTF−8 smiley face
Db' u263a'                   ; UTF-8 笑脸
db
数
据  '\xe2\x98\xba'              UTF−8  smiley   face
库  Xe2 x98 xba               ;  UTF-8  笑脸     笑脸
db
数  0E2h, 098h, 0BAh            UTF−8  smiley   face
据  0E2h 098h 0BAh            ;  UTF-8  笑脸     笑脸
```

库

3.4.3 Character Constants
3.4.3 字符常数

A character constant consists of a string up to eight bytes long, used in an expression context. It is treated as if it was an integer.
一个字符常量由一个长达 8 字节的字符串组成，用于表达式上下文中。它被当作一个整数来对待。

A character constant with more than one byte will be arranged with little−endian order in mind: if you code
如果您编写代码，那么一个字节以上的字符常量将按照小末尾顺序进行安排

```
mov eax,'abcd'
Mov eax,'abcd'
```

then the constant generated is not `0x61626364`, but `0x64636261`, so that if you were then to store the value into memory, it would read `abcd` rather than `dcba`. This is also the sense of character constants understood by the Pentium's `CPUID` instruction.
那么生成的常量就不是 0x61626364 而是 0x64636261，所以如果要将值存储到内存中，它将读取 abcd 而不是 dcba。这也是 Pentium 的 CPUID 指令所理解的字符常量的含义。

3.4.4 String Constants
3.4.4 字符串常量

String constants are character strings used in the context of some pseudo−instructions, namely the `DB` family and `INCBIN` (where it represents a filename.) They are also used in certain preprocessor directives.
String 常量是在一些伪指令上下文中使用的字符串，即 DB 家族和 INCBIN (其中它表示文件名)它们也在某些预处理器指令中使用。

A string constant looks like a character constant, only longer. It is treated as a concatenation of maximum−size character constants for the conditions. So the following are equivalent:
一个字符串常量看起来像一个字符常量，只是更长。它被看作是这些条件的最大字符常量的串联。所以下面是等价的:

```
db    'hello'               ; string constant
Db'hello'; 字符串常量
db    'h','e','l','l','o'   ; equivalent character constants
Db'h','e','l','l','o'; 等效字符常量
```

And the following are also equivalent:
下面这些也是等价的:

```
dd
编
译:
pest   'ninechars'             ; doubleword     string constant
wave   九个人                  双关语           字符串常量
dd
编
译:
pest   'nine','char','s'         becomes three doublewords
wave   '9'' char's'            ;  变成了三个双关语
db
数据   'ninechars',0,0,0         and really     looks like this
库     " ninechars"0000        ;  真的          看起来像这样
```

Note that when used in a string−supporting context, quoted strings are treated as a string constants even if they are short enough to be a character constant, because otherwise `db 'ab'` would have the

注意，当在支持字符串的上下文中使用时，即使被引用的字符串短到足以成为字符常量，也会被视为字符串常量，因为否则 db’ab’将具有

36

same effect as `db 'a'`, which would be silly. Similarly, three−character or four−character constants are treated as strings when they are operands to `DW`, and so forth.
和 db'a'一样的效果，那就太傻了。类似地，三个字符或四个字符的常量在作为 DW 的操作数时被视为字符串，等等。

3.4.5 Unicode Strings
3.4.5 Unicode 字符串

The special operators `__utf16__`, `__utf16le__`, `__utf16be__`, `__utf32__`, `__utf32le__` and `__utf32be__` allows definition of Unicode strings. They take a string in UTF−8 format and converts it to UTF−16 or UTF−32, respectively. Unless the `be` forms are specified, the output is littleendian.
特殊运算符 _ utf16 _ _、 _ utf16le _、 _ utf16be _、 _ utf32 _、 _ utf32le _ 和 _ utf32be _ 允许定义 Unicode 字符串。它们接受 UTF-8 格式的字符串，并分别将其转换为 UTF-16 或 UTF-32。除非指定 be 格式，否则输出是 littleendian。

For example:
例如:

```
%define u(x) __utf16__(x)
% 定义 u (x) _ utf16 _ (x)
%define w(x) __utf32__(x)
% 定义 w (x) _ utf32 _ _ (x)

      dw u('C:\WINDOWS'), 0       ; Pathname in UTF-16
      Dw u ('c: WINDOWS') , 0; UTF-16 中的 Pathname
      dd w(`A + B = \u206a`), 0   ; String in UTF-32
      Dd w (` a + b = u206a') , 0; UTF-32 中的字符串
```

The UTF operators can be applied either to strings passed to the `DB` family instructions, or to character constants in an expression context.
UTF 运算符既可以应用于传递给 DB 家族指令的字符串，也可以应用于表达式上下文中的字符常量。

3.4.6 Floating−Point Constants
3.4.6 Floating-Point 常量

Floating−point constants are acceptable only as arguments to `DB`, `DW`, `DD`, `DQ`, `DT`, and `DO`, or as arguments to the special operators `__float8__`, `__float16__`, `__float32__`, `__float64__`, `__float80m__`, `__float80e__`, `__float128l__`, and `__float128h__`.
浮点常数只能作为 DB、 DW、 DD、 DQ、 DT 和 DO 的参数，或作为特殊运算符 _ _ float8 _、 _ _ float16 _、 _ _ float32 _、 _ _ float64 _、 _ _ float80m _、 _ _ float80e _、 _ _ float128l _ 和 _ float128h _ _ 的参数。

Floating−point constants are expressed in the traditional form: digits, then a period, then optionally more digits, then optionally an `E` followed by an exponent. The period is mandatory, so that NASM can distinguish between `dd 1`, which declares an integer constant, and `dd 1.0` which declares a floating−point constant.
浮点常数以传统形式表示: 先是数字，然后是句点，然后是可选的更多数字，然后是可选的 e，最后是指数。句点是强制性的，因此 NASM 可以区分声明整数常量的 dd 1 和声明浮点常量的 dd 1.0。

NASM also support C99−style hexadecimal floating−point: `0x`, hexadecimal digits, period, optionally more hexadeximal digits, then optionally a `P` followed by a *binary* (not hexadecimal) exponent in decimal notation. As an extension, NASM additionally supports the `0h` and `$` prefixes for hexadecimal, as well binary and octal floating−point, using the `0b` or `0y` and `0o` or `0q` prefixes, respectively.
NASM 还支持 c99 风格的十六进制浮点数: 0x，十六进制数字，周期，可选的更多十六进制数字，然后可选的 p 后面是十进制表示法中的二进制(非十六进制)指数。作为一个扩展，NASM 还支持分别使用 0b 或 0y 和 0o 或 0q 前缀的 16 进制、二进制和八进制浮点数的 0h 和 $前缀。

Underscores to break up groups of digits are permitted in floating−point constants as well.

在浮点常量中也允许使用下划线来分解数字组。

Some examples:
一些例子:

```
db
数据   −0.2                        ; "Quarter precision"
库     0.2                         ; 「季度精度」
dw     −0.5                        ; IEEE 754r/SSE5 half precision
Dw     -0.5                        ; IEEE 754r/sse5 半精度
dd
编
译:
pest                               ; an easy one
wave   1.2                         一个简单的
dd
编
译:
pest   1.222_222_222               ; underscores are permitted
wave   1.222 _ 222 _ 222           下划线是允许的
dd
编
译:
pest   0x1p+2                      ; 1.0x2^2 = 4.0
wave   0x1p + 2                    ; 1.0 x2 ^ 2 = 4.0
dq     0x1p+32                     ; 1.0x2^32 = 4 294 967 296.0
Dq     0x1p + 32                   ; 1.0 x2 ^ 32 = 4294967296.0
dq     1.e10                       ; 10 000 000 000.0
Dq     第一季，第 10 集            ; 10000000000
dq     1.e+10                      ; synonymous with 1.e10
Dq     1. e + 10                   ; 等同于 1. e10
dq     1.e−10                      ; 0.000 000 000 1
Dq     1. e-10                     ; 0.000000000
dt                                 ; pi
Dt     3.141592653589793238462     圆周率
do     1.e+4000                    ; IEEE 754r quad precision
做     1. e + 4000                 IEEE 754r 四精度
```

The 8−bit "quarter−precision" floating−point format is sign:exponent:mantissa = 1:4:3 with an exponent bias of 7. This appears to be the most frequently used 8−bit floating−point format, although it is not covered by any formal standard. This is sometimes called a "minifloat."
8 位“四分之一精度”浮点格式是符号: 指数: 尾数 = 1:4:3，指数偏差为 7。这似乎是最常用的 8 位浮点格式，虽然它没有被任何正式的标准覆盖。这有时被称为“迷你浮点数”

The special operators are used to produce floating−point numbers in other contexts. They produce the binary representation of a specific floating−point number as an integer, and can use anywhere integer constants are used in an expression. `__float80m__` and `__float80e__` produce the 64−bit
这些特殊的运算符用于在其他上下文中产生浮点数。它们以整数的形式产生特定浮点数的二进制表示，并且可以在表达式中使用整数常量的任何位置。_ _ float80m _ _ 和 _ _ float80e _ _ 生成 64 位

37

mantissa and 16-bit exponent of an 80-bit floating-point number, and `__float128l__` and `__float128h__` produce the lower and upper 64-bit halves of a 128-bit floating-point number, respectively.
80 位浮点数的尾数和 16 位指数，以及 _ float128l _ 和 _ float128h _ 分别产生 128 位浮点数的下 64 位和上 64 位的一半。

For example:
例如:

```
      mov     rax,__float64__(3.141592653589793238462)
      Mov rax， _ float64 _ (3.141592653589793238462)
```

... would assign the binary representation of pi as a 64-bit floating point number into `RAX`. This is exactly equivalent to:
将 pi 的二进制表示形式作为 64 位浮点数分配到 RAX 中。这完全等价于:

```
      mov     rax,0x400921fb54442d18
      Mov rax，0 x400921fb54442d18
```

NASM cannot do compile-time arithmetic on floating-point constants. This is because NASM is designed to be portable – although it always generates code to run on x86 processors, the assembler itself can run on any system with an ANSI C compiler. Therefore, the assembler cannot guarantee the presence of a floating-point unit capable of handling the Intel number formats, and so for NASM to be able to do floating arithmetic it would have to include its own complete set of floating-point routines, which would significantly increase the size of the assembler for very little benefit.
NASM 不能对浮点常量进行编译时算术。这是因为 NASM 被设计成可移植的——尽管它总是生成在 x86 处理器上运行的代码，但是汇编程序本身可以在任何使用 ANSI c 编译器的系统上运行。因此，汇编程序不能保证存在一个能够处理英特尔数字格式的浮点单元，因此，为了使 NASM 能够进行浮点运算，它必须包括自己的一套完整的浮点例程，这将大大增加汇编程序的大小，但收益很小。

The special tokens `__Infinity__`, `__QNaN__` (or `__NaN__`) and `__SNaN__` can be used to generate infinities, quiet NaNs, and signalling NaNs, respectively. These are normally used as macros:
特殊标记 _ Infinity _、 _ QNaN _ (或 _ NaN _)和 _ SNaN _ 可以分别用于生成无穷大、安静的 NaN 和信号 NaN。这些通常用作宏:

```
%define Inf __Infinity__
% 定义 Inf _ _ Infinity _ _
%define NaN __QNaN__
% 定义 NaN _ QNaN _

      dq    +1.5, -Inf, NaN          ; Double-precision constants
      Dq + 1.5,-Inf，NaN; 双精度常数
```

The `%use fp` standard macro package contains a set of convenience macros. See section 5.3.
% use fp 标准宏包含一组方便的宏。参见第 5.3 节。

3.4.7 Packed BCD Constants
3.4.7 包装 BCD 常数

x87-style packed BCD constants can be used in the same contexts as 80-bit floating-point numbers.
X87 样式的压缩 BCD 常量可以在与 80 位浮点数相同的上下文中使用。
They are suffixed with `p` or prefixed with `0p`, and can include up to 18 decimal digits.
它们以 p 为后缀或以 0p 为前缀，最多可包括 18 个小数位。

As with other numeric constants, underscores can be used to separate digits.
和其他数字常量一样，下划线可以用来分隔数字。

For example:
例如:

```
      dt 12_345_678_901_245_678p
      Dt 12 _ 345 _ 678 _ 901 _ 245 _ 678p
      dt -12_345_678_901_245_678p
```

```
Dt-12 _ 345 _ 678 _ 901 _ 245 _ 678p
dt +0p33
Dt + 0 p33
dt 33p
Dt33p
```

3.5 Expressions

3.5 表达式

Expressions in NASM are similar in syntax to those in C. Expressions are evaluated as 64−bit integers which are then adjusted to the appropriate size.
NASM 中的表达式在语法上与 c 中的表达式相似。表达式计算为 64 位整数，然后调整到适当的大小。

NASM supports two special tokens in expressions, allowing calculations to involve the current assembly position: the `$` and `$$` tokens. `$` evaluates to the assembly position at the beginning of the line containing the expression; so you can code an infinite loop using `JMP $`. `$$` evaluates to the beginning of the current section; so you can tell how far into the section you are by using `($-$$)`.
NASM 支持表达式中的两个特殊标记，允许计算包含当前的汇编位置: $和 $标记。$计算到包含表达式的行的开头的汇编位置; 所以你可以使用 JMP $编写一个无限循环。$$计算到当前部分的开头; 因此可以通过使用($- $)来判断进入该部分的距离。

The arithmetic operators provided by NASM are listed here, in increasing order of precedence.
NASM 提供的算术运算符在这里按优先顺序递增列出。

3.5.1 `|`: Bitwise OR Operator

3.5.1 | : Bitwise OR 运算符

The `|` operator gives a bitwise OR, exactly as performed by the `OR` machine instruction. Bitwise OR is the lowest−priority arithmetic operator supported by NASM.
| 运算符给出一个按位 OR，完全按照 OR 机器指令执行。Bitwise OR 是 NASM 支持的最低优先级算术运算符。

3.5.2 ^: Bitwise XOR Operator
3.5.2 ^ : Bitwise XOR 运算符

`^` provides the bitwise XOR operation.
提供按位 XOR 操作。

3.5.3 &: Bitwise AND Operator
3.5.3 & : Bitwise AND Operator

`&` provides the bitwise AND operation.
提供按位 AND 运算。

3.5.4 << and >>: Bit Shift Operators
3.5.4 < < 和 > > : Bit Shift 操作符

`<<` gives a bit−shift to the left, just as it does in C. So `5<<3` evaluates to 5 times 8, or 40. `>>` gives a bit−shift to the right; in NASM, such a shift is *always* unsigned, so that the bits shifted in from the left−hand end are filled with zero rather than a sign−extension of the previous highest bit.
给出一个位移到左边，就像在 c 中一样。所以 5 < 3 等于 5 乘以 8，或者 40。> > 向右移位; 在 NASM 中，这种移位总是无符号的，因此从左端移入的位填充的是零，而不是前一个最高位的符号扩展。

3.5.5 + and −: Addition and Subtraction Operators
3.5.5 + 和-: 加减运算符

The `+` and `−` operators do perfectly ordinary addition and subtraction.
+ 和-运算符完全可以做普通的加减运算。

3.5.6 *, /, //, % and %%: Multiplication and Division
3.5.6 * ,/,//,% 和% : 乘法和除法

`*` is the multiplication operator. `/` and `//` are both division operators: `/` is unsigned division and `//` is signed division. Similarly, `%` and `%%` provide unsigned and signed modulo operators respectively.
是乘法运算符。/和/都是除法运算符:/是无符号除法和//是有符号除法。同样,% 和% 分别提供了无符号和有符号的模运算符。

NASM, like ANSI C, provides no guarantees about the sensible operation of the signed modulo operator.
NASM 与 ANSI c 一样，不能保证有符号模运算符的合理操作。

Since the `%` character is used extensively by the macro preprocessor, you should ensure that both the signed and unsigned modulo operators are followed by white space wherever they appear.
由于% 字符被宏预处理器广泛使用，您应该确保有符号和无符号的模运算符后面都有空格，无论它们出现在哪里。

3.5.7 Unary Operators
3.5.7 一元运算符

The highest−priority operators in NASM's expression grammar are those which only apply to one argument. These are `+`, `−`, `~`, `!`, `SEG`, and the integer functions operators.
NASM 表达式语法中最高优先级的运算符是那些只适用于一个参数的运算符。它们是 + ,-，~ ！，SEG，和整数函数操作符。

`−` negates its operand, `+` does nothing (it's provided for symmetry with `−`), `~` computes the one's complement of its operand, `!` is the logical negation operator.
否定它的操作数，+ 什么也不做(它提供了与-的对称性) ，~ 计算它的操作数的补码，！ 是逻辑非运算符。

`SEG` provides the segment address of its operand (explained in more detail in section 3.6).
`SEG` 提供了操作数的段地址(详见第 3.6 节)。

A set of additional operators with leading and trailing double underscores are used to implement the integer functions of the `ifunc` macro package, see section 5.4.
一组带有前后双下划线的附加运算符用于实现 ifunc 宏包的整数函数，参见第 5.4 节。

3.6 SEG and WRT
3.6 SEG 和 WRT

When writing large 16−bit programs, which must be split into multiple segments, it is often necessary to be able to refer to the segment part of the address of a symbol. NASM supports the `SEG` operator to perform this function.
当编写大型 16 位程序时，必须将其分成多个段，通常需要能够引用符号地址的段部分。NASM 支持 SEG 操作符来执行这个功能。

The `SEG` operator returns the *preferred* segment base of a symbol, defined as the segment base relative to which the offset of the symbol makes sense. So the code
SEG 操作符返回符号的首选段基，定义为相对于符号偏移量有意义的段基。所以代码

```
mov     ax,seg symbol
Mov ax 禁闭室符号
mov     es,ax
动起来，斧头
mov     bx,symbol
Mov bx 符号
```

will load `ES:BX` with a valid pointer to the symbol `symbol`.
将使用指向符号符号的有效指针加载 ES: BX。

Things can be more complex than this: since 16−bit segments and groups may overlap, you might occasionally want to refer to some symbol using a different segment base from the preferred one. NASM lets you do this, by the use of the `WRT` (With Reference To) keyword. So you can do things like
事情可能比这更复杂: 由于 16 位段和组可能重叠，您可能偶尔想要使用与首选段基不同的段基来引用某些符号。NASM 允许你这样做，通过使用 WRT (With Reference To)关键字。所以你可以这样做

```
mov     ax,weird_seg        ; weird_seg is a segment base
Mov ax，weird _ seg; weird _ seg 是段基
mov     es,ax
动起来，斧头
mov     bx,symbol wrt weird_seg
Mov bx，symbol wrt weird _ seg
```

to load `ES:BX` with a different, but functionally equivalent, pointer to the symbol `symbol`.
使用一个不同的，但功能等同的，指向符号符号的指针来加载 ES: BX。

NASM supports far (inter−segment) calls and jumps by means of the syntax `call segment:offset`, where `segment` and `offset` both represent immediate values. So to call a far procedure, you could code either of
NASM 通过语法调用段: offset 支持远程(段间)调用和跳转，其中段和偏移量都表示即时值。所以要调用一个 far 过程，你可以使用

```
call    (seg procedure):procedure
调用(seg 过程) : 过程
call    weird_seg:(procedure wrt weird_seg)
调用 weird _ seg: (程序 wrt weird _ seg)
```

(The parentheses are included for clarity, to show the intended parsing of the above instructions. They are not necessary in practice.)
(为了清晰起见，括号中包含了对上述指令的解析。它们在实践中是不必要的

NASM supports the syntax `call far procedure` as a synonym for the first of the above usages.
NASM 支持将语法调用 far 过程作为上述第一种用法的同义词。
`JMP` works identically to `CALL` in these examples.
在这些例子中，JMP 和 CALL 的工作原理是一样的。

To declare a far pointer to a data item in a data segment, you must code
若要声明指向数据段中的数据项的远程指针，必须编写代码

```
dw      symbol, seg symbol
Dw 符号，seg 符号
```

NASM supports no convenient synonym for this, though you can always invent one using the macro processor.
NASM 不支持这种方便的同义词，尽管您总是可以使用宏处理器创建一个。

3.7 `STRICT`: Inhibiting Optimization

3.7 STRICT: 抑制优化

When assembling with the optimizer set to level 2 or higher (see section 2.1.23), NASM will use size specifiers (`BYTE`, `WORD`, `DWORD`, `QWORD`, `TWORD`, `OWORD`, `YWORD` or `ZWORD`), but will give them the smallest possible size. The keyword `STRICT` can be used to inhibit optimization and force a particular operand to be emitted in the specified size. For example, with the optimizer on, and in `BITS 16` mode,
当将优化器设置为 2 级或更高级别(参见 2.1.23 节)时，NASM 将使用大小说明符(BYTE、 WORD、 DWORD、 QWORD、 TWORD、 OWORD、 YWORD 或 ZWORD) ，但会给它们尽可能小的大小。关键字 STRICT 可以用来抑制优化并强制一个特定的操作数以指定的大小发出。例如，使用优化器，在 BITS 16 模式下,

```
push dword 33
按 dword 33
```

is encoded in three bytes `66 6A 21`, whereas
编码为三个字节 666a21，而

```
push strict dword 33
推严格的 dword 33
```

is encoded in six bytes, with a full dword immediate operand `66 68 21 00 00 00`.
以六个字节编码，带有一个完整的 dword immediate 操作数 666821000000。

With the optimizer off, the same code (six bytes) is generated whether the `STRICT` keyword was used or not.
关闭优化器后，不管是否使用 STRICT 关键字，都会生成相同的代码(6 字节)。

3.8 Critical Expressions

3.8 关键表达式

Although NASM has an optional multi−pass optimizer, there are some expressions which must be resolvable on the first pass. These are called *Critical Expressions*.
虽然 NASM 有一个可选的多通道优化器，但是有一些表达式必须在第一次通过时解析。这些被称为关键表达式。

The first pass is used to determine the size of all the assembled code and data, so that the second pass, when generating all the code, knows all the symbol addresses the code refers to. So one thing NASM can't handle is code whose size depends on the value of a symbol declared after the code in question. For example,
第一遍用于确定所有汇编代码和数据的大小，因此第二遍在生成所有代码时知道代码引用的所有符号地址。所以 NASM 不能处理的一件事就是代码的大小取决于在代码后面声明的符号的值。例如,

```
        times (label-$) db 0
        times (label-$) db 0
label:  db      'Where am I?'
标签: db'我在哪里?'
```

The argument to `TIMES` in this case could equally legally evaluate to anything at all; NASM will reject this example because it cannot tell the size of the `TIMES` line when it first sees it. It will just as firmly reject the slightly paradoxical code
在这种情况下，对 TIMES 的论证可以同样合法地评估任何东西; NASM 将拒绝这个例子，因为它不能告诉时代线的大小，当它第一次看到它。它也会坚决拒绝这些略显矛盾的代码

```
        times (label-$+1) db 0
        乘以(标签-$+ 1) db 0
label:  db      'NOW where am I?'
标签: db'现在我在哪里
```

in which *any* value for the `TIMES` argument is by definition wrong!
根据定义，TIMES 的任何值都是错误的!

NASM rejects these examples by means of a concept called a *critical expression*, which is defined to be an expression whose value is required to be computable in the first pass, and which must therefore depend only on symbols defined before it. The argument to the `TIMES` prefix is a critical expression.
NASM 通过一个称为临界表达式的概念来拒绝这些示例，临界表达式被定义为其值必须在第一次通过时可计算的表达式，因此必须仅依赖于在其之前定义的符号。TIMES 前缀的参数是一个临界表达式。

3.9 Local Labels
3.9 本地标签

NASM gives special treatment to symbols beginning with a period. A label beginning with a single period is treated as a *local* label, which means that it is associated with the previous non−local label. So, for example:
NASM 对以句号开头的符号给予特殊处理。以单个句点开头的标签被视为本地标签，这意味着它与之前的非本地标签相关联。例如:

```
label1  ; some code
一些代码

.loop
循环
        ; some more code
        更多的代码

        jne     .loop
        Jne 循环
        ret
        后悔

label2  ; some code
一些代码

.loop
循环
        ; some more code
        更多的代码

        jne     .loop
        Jne 循环
        ret
        后悔
```

In the above code fragment, each `JNE` instruction jumps to the line immediately before it, because the two definitions of `.loop` are kept separate by virtue of each being associated with the previous non−local label.
在上面的代码片段中，每条 JNE 指令都跳转到它前面的行，因为。循环是分开的，因为每个循环都与之前的非本地标签相关联。

This form of local label handling is borrowed from the old Amiga assembler DevPac; however, NASM goes one step further, in allowing access to local labels from other parts of the code. This is achieved by means of *defining* a local label in terms of the previous non−local label: the first definition of `.loop` above is really defining a symbol called `label1.loop`, and the second defines a symbol called `label2.loop`. So, if you really needed to, you could write
这种形式的本地标签处理借鉴了老的 Amiga 汇编程序 DevPac; 然而，NASM 更进一步，允许从代码的其他部分访问本地标签。这是通过以前的非本地标签定义本地标签来实现的: 第一个定义。上面的循环实际上定义了一个名为 label1.loop 的符号，第二个定义了一个名为 label2.loop 的符号。所以，如果你真的需要，你可以写

```
label3  ; some more code
一些更多的代码
```

```
        ; and some more
        还有更多

        jmp label1.loop
        Jmp label1.loop
```

Sometimes it is useful – in a macro, for instance – to be able to define a label which can be referenced from anywhere but which doesn’t interfere with the normal local−label mechanism. Such a label can’t be non−local because it would interfere with subsequent definitions of, and references to, local labels; and it can’t be local because the macro that defined it wouldn’t know the label’s full name. NASM therefore introduces a third type of label, which is probably only useful in macro definitions: if a label begins with the special prefix `..@`, then it does nothing to the local label mechanism. So you could code
有时它是有用的——例如在宏中——能够定义一个标签，这个标签可以在任何地方被引用，但是不会干扰正常的本地标签机制。这样的标签不能是非本地的，因为它会干扰本地标签的后续定义和对本地标签的引用; 它不能是本地的，因为定义它的宏不会知道标签的全名。因此，NASM 引入了第三种类型的标签，这可能只在宏定义中有用: 如果一个标签以特殊的前缀开头。.@，那么它对本地标签机制没有任何作用。所以你可以编码

```
label1:                         ; a non-local label
一个非本地标签
.local:                         ; this is really label1.local
这真的是 label1.local
..@foo:                         ; this is a special symbol
@ foo: ; 这是一个特殊的符号
label2:                         ; another non-local label
标签 2: ; 另一个非本地标签
.local:                         ; this is really label2.local
Local: ; 这实际上是 label2.local

        jmp     ..@foo          ; this will jump three lines up
        @ foo; 这将跳转三行
```

NASM has the capacity to define other special symbols beginning with a double period: for example, ..start is used to specify the entry point in the obj output format (see section 7.4.6),
NASM 有能力定义其他以双周期开头的特殊符号: 例如,。.Start 用于在对象输出格式中指定入口点(参见第 7.4.6 节),

`..imagebase` is used to find out the offset from a base address of the current image in the `win64` output format (see section 7.6.1). So just keep in mind that symbols beginning with a double period are special.
`..Imagebase` 用于以 `win64` 输出格式查找当前图像的基地址的偏移量(参见 7.6.1 节)。所以请记住，以双周期开头的符号是特殊的。

Chapter 4: The NASM Preprocessor
第四章: NASM 预处理器

NASM contains a powerful macro processor, which supports conditional assembly, multi−level file inclusion, two forms of macro (single−line and multi−line), and a 'context stack' mechanism for extra macro power. Preprocessor directives all begin with a `%` sign.
NASM 包含一个强大的宏处理器，它支持条件汇编、多级文件包含、两种形式的宏(单行和多行)，以及一个"上下文堆栈"机制来提供额外的宏功能。预处理器指令都以% 符号开头。

The preprocessor collapses all lines which end with a backslash (\) character into a single line. Thus:
预处理器将所有以反斜杠()结尾的行折叠成一行。如下:

```
%define THIS_VERY_LONG_MACRO_NAME_IS_DEFINED_TO \
% 定义这个 _ very _ long _ macro _ name _ is _ defined _ to
        THIS_VALUE
        这个 _ 值
```

will work like a single−line macro without the backslash−newline sequence.
将像一个没有反斜杠-换行符序列的单行宏一样工作。

4.1 Single−Line Macros
4.1 单行宏

4.1.1 The Normal Way: `%define`
4.1.1 Normal Way:% define

Single−line macros are defined using the `%define` preprocessor directive. The definitions work in a similar way to C; so you can do things like
单行宏使用% define 预处理器指令定义。定义的工作方式类似于 c，所以你可以这样做

```
%define
百分比定 ctrl       0x1F &
义       Ctrl       0x1F &
%define
百分比定 param(a,b) ((a)+(a)*(b))
义       (a) + (a) * (b))
         mov        byte   [param(2,ebx)], ctrl 'D'
         动起来     字节   [ param (2，ebx)]，ctrl'd'
```

which will expand to
它会扩展到

```
         mov        byte   [(2)+(2)*(ebx)], 0x1F & 'D'
         动起来     字节   [(2) + (2) * (ebx)]，0x1F &'d'
```

When the expansion of a single−line macro contains tokens which invoke another macro, the expansion is performed at invocation time, not at definition time. Thus the code
当单行宏的展开包含调用另一个宏的令牌时，展开在调用时执行，而不是在定义时执行。因此，代码

```
%define a(x)    1+b(x)
% 定义 a (x)1 + b (x)
%define b(x)    2*x
% 定义 b (x)2 * x

        mov     ax,a(8)
        Mov ax，a (8)
```

will evaluate in the expected way to `mov ax,1+2*8`, even though the macro `b` wasn't defined at the time of definition of `a`.
将以预期的方式计算 mov ax，1 + 2 * 8，即使宏 b 在定义 a 时没有定义。

Macros defined with `%define` are case sensitive: after `%define foo bar`, only `foo` will expand to `bar`: `Foo` or `FOO` will not. By using `%idefine` instead of `%define` (the 'i' stands for 'insensitive') you can define all the case variants of a macro at once, so that `%idefine foo bar` would cause `foo`, `Foo`, `FOO`, `fOO` and so on all to expand to `bar`.
用% define 定义的宏是区分大小写的: 在% define FOO bar 之后，只有 FOO 会扩展到 bar: FOO 或 FOO 不会。通过使用% idedefine 而不是% define (' i'代表' insensitive')，您可以一次性定义宏的所有大小写变体，这样% idedefine FOO bar 将导致 FOO、 FOO、 FOO、 FOO 等全部扩展为 bar。

There is a mechanism which detects when a macro call has occurred as a result of a previous expansion of the same macro, to guard against circular references and infinite loops. If this happens, the preprocessor will only expand the first occurrence of the macro. Hence, if you code
有一种机制可以检测宏调用何时由于同一宏的先前扩展而发生，以防止循环引用和无限循环。如果发生这种情况，预处理器将只展开宏的第一次出现。因此，如果你编码

```
%define a(x)    1+a(x)
定义 a (x)1 + a (x)

        mov     ax,a(3)
        Mov ax, a (
```

the macro `a(3)` will expand once, becoming `1+a(3)`, and will then expand no further. This behaviour can be useful: see section 9.1 for an example of its use.
宏 a (3)将扩展一次，变成 1 + a (3)，然后不再扩展。这种行为可能是有用的: 参见第 9.1 节的一个使用例子。

You can overload single−line macros: if you write
您可以重载单行宏: 如果您编写

```
%define foo(x)   1+x
% 定义 foo (x)1 + x
%define foo(x,y) 1+x*y
% 定义 foo (x, y)1 + x * y
```

43

the preprocessor will be able to handle both types of macro call, by counting the parameters you pass; so `foo(3)` will become `1+3` whereas `foo(ebx,2)` will become `1+ebx*2`. However, if you define
预处理器将能够处理这两种类型的宏调用，通过计算您传递的参数; 因此 foo (3)将成为 1 + 3，而 foo (ebx，2)将成为 1 + ebx * 2。然而，如果你定义

```
%define foo bar
% 定义 foo bar
```

then no other definition of `foo` will be accepted: a macro with no parameters prohibits the definition of the same name as a macro *with* parameters, and vice versa.
那么 foo 的其他定义将不被接受: 没有参数的宏不允许定义与带参数的宏相同的名称，反之亦然。

This doesn't prevent single−line macros being *redefined*: you can perfectly well define a macro with
这并不能阻止单行宏被重新定义: 你完全可以用

```
%define foo bar
% 定义 foo bar
```

and then re−define it later in the same source file with
然后在同一源文件中重新定义它

```
%define foo baz
% 定义 foo baz
```

Then everywhere the macro `foo` is invoked, it will be expanded according to the most recent definition. This is particularly useful when defining single−line macros with `%assign` (see section 4.1.7).
然后在宏 foo 被调用的任何地方，它都会根据最新的定义进行扩展。这在定义带有% assign 的单行宏时特别有用(参见第 4.1.7 节)。

You can pre−define single−line macros using the '−d' option on the NASM command line: see section 2.1.19.
您可以使用 NASM 命令行上的"-d"选项预先定义单行宏: 参见第 2.1.19 节。

4.1.2 Resolving `%define`: `%xdefine`
4.1.2 resolution% define:% xdefine

To have a reference to an embedded single−line macro resolved at the time that the embedding macro is *defined*, as opposed to when the embedding macro is *expanded*, you need a different mechanism to the one offered by `%define`. The solution is to use `%xdefine`, or it's case−insensitive counterpart `%ixdefine`.
为了在定义嵌入宏时解析对嵌入单行宏的引用，而不是在扩展嵌入宏时解析引用，您需要与% define 提供的机制不同的机制。解决方案是使用% xdefine，或者不区分大小写的对应方法% ixdefine。

Suppose you have the following code:
假设你有以下代码:

```
%define  isTrue  1
% 定义 isTrue 1
%define  isFalse isTrue
% 定义 isFalse isTrue
%define  isTrue  0
% 定义 isTrue 0
```

```
val1:       db          isFalse
第一步:     数据库      是假的
%define     isTrue
百分比定    伊斯特
义          鲁          1

val2:       db          isFalse
```

Val2:	数据库	是假的

In this case, `val1` is equal to 0, and `val2` is equal to 1. This is because, when a single−line macro is defined using `%define`, it is expanded only when it is called. As `isFalse` expands to `isTrue`, the expansion will be the current value of `isTrue`. The first time it is called that is 0, and the second time it is 1.
在这种情况下，val1 等于 0，val2 等于 1。这是因为，当一个单行宏使用% define 定义时，它只有在被调用时才被展开。当 isFalse 扩展到 isTrue 时，这个扩展就是 isTrue 的当前值。第一次调用时为 0，第二次调用时为 1。

If you wanted `isFalse` to expand to the value assigned to the embedded macro `isTrue` at the time that `isFalse` was defined, you need to change the above code to use `%xdefine`.
如果希望 isFalse 扩展到定义 isFalse 时分配给嵌入式宏 isTrue 的值，则需要将上述代码更改为使用% xdefine。

```
%xdefine isTrue  1
% xdefine isTrue 1
%xdefine isFalse isTrue
% xdefine isFalse isTrue
%xdefine isTrue  0
% xdefine isTrue 0
```

val1:	db	isFalse
第一步:	数据库	是假的
	isTrue	
%xdefine	伊斯特	
% xdefine	鲁	1
val2:	db	isFalse
Val2:	数据库	是假的

Now, each time that `isFalse` is called, it expands to 1, as that is what the embedded macro `isTrue` expanded to at the time that `isFalse` was defined.
现在，每次调用 isFalse 时，它都会扩展到 1，因为在定义 isFalse 时，嵌入式宏 isTrue 扩展到 1。

4.1.3 Macro Indirection: `%[ ... ]`
4.1.3 宏间接:% [...]

The `%[...]` construct can be used to expand macros in contexts where macro expansion would otherwise not occur, including in the names other macros. For example, if you have a set of macros named `Foo16`, `Foo32` and `Foo64`, you could write:
% [...]构造可以用来在宏不会展开的上下文中展开宏，包括其他宏的名称。例如，如果你有一组名为 Foo16，foo32 和 foo64 的宏，你可以写:

```
     mov ax,Foo%[__BITS__]    ; The Foo value
     Foo% [ _ _ BITS _ ] ; Foo 值
```

to use the builtin macro `__BITS__` (see section 4.11.5) to automatically select between them.
使用内置宏 _ _ BITS _ (参见第 4.11.5 节)自动在它们之间进行选择。
Similarly, the two statements:
同样，这两个陈述:

```
%xdefine Bar         Quux    ; Expands due to %xdefine
% xdefine Bar Quux; expand due to% xdefine
%define  Bar         %[Quux] ; Expands due to %[...]
% 定义 Bar% [ Quux ] ; 由于% [ ... ]而扩展
```

have, in fact, exactly the same effect.
实际上，具有完全相同的效果。

`%[...]` concatenates to adjacent tokens in the same way that multi−line macro parameters do, see section 4.3.9 for details.
% [...]以与多行宏参数相同的方式连接到相邻的令牌，有关详细信息，请参阅第 4.3.9 节。

4.1.4 Concatenating Single Line Macro Tokens: `%+`
4.1.4 连接单行宏令牌:% +

Individual tokens in single line macros can be concatenated, to produce longer tokens for later processing. This can be useful if there are several similar macros that perform similar functions.
单行宏中的单个令牌可以连接起来，以产生更长的令牌供以后处理。如果有几个相似的宏执行相似的功能，这会很有用。

Please note that a space is required after `%+`, in order to disambiguate it from the syntax `%+1` used in multiline macros.
请注意，在% + 之后需要一个空格，以消除它与多行宏中使用的语法% + 1 的歧义。

As an example, consider the following:
作为一个例子，考虑以下几点:

```
%define BDASTART 400h                ; Start of BIOS data area
% 定义 BDASTART 400h; 开始 BIOS 数据区

struc   tBIOSDA                      ; its structure
结构; 其结构
        .COM1addr       RESW    1
        COM1addr RESW 1
        .COM2addr       RESW    1
        . COM2addr RESW 1
        ; ..and so on
        等等
endstruc
内部结构
```

Now, if we need to access the elements of tBIOSDA in different places, we can end up with:
现在，如果我们需要在不同的地方访问 tBIOSDA 的元素，我们可以得到:

```
        mov     ax,BDASTART + tBIOSDA.COM1addr
        BDASTART + tbiosda.com 1addr
        mov     bx,BDASTART + tBIOSDA.COM2addr
        Mov bx，BDASTART + tBIOSDA.COM2addr
```

This will become pretty ugly (and tedious) if used in many places, and can be reduced in size significantly by using the following macro:
如果在许多地方使用，这将变得相当丑陋(和乏味)，并且可以通过使用以下宏来显著减小尺寸:

```
; Macro to access BIOS variables by their names (from

tBDA): %define BDA(x) BDASTART + tBIOSDA. %+ x

宏通过名称访问 BIOS 变量(来自 tBDA) :% 定义 BDA (x) BDASTART +

tBIOSDA。% + x
```

Now the above code can be written as:
现在，上面的代码可以写成:

```
        mov     ax,BDA(COM1addr)
        Mov ax，BDA (COM1addr)
        mov     bx,BDA(COM2addr)
        电影，BDA (COM2addr)
```

Using this feature, we can simplify references to a lot of macros (and, in turn, reduce typing errors).
使用这个特性，我们可以简化对许多宏的引用(进而减少输入错误)。

4.1.5 The Macro Name Itself: `%?` and `%??`
宏名称本身:% ? 和% ?

The special symbols `%?` and `%??` can be used to reference the macro name itself inside a macro expansion, this is supported for both single−and multi−line macros. `%?` refers to the macro name as
特殊符号% ? 和% ? 可以用来引用宏展开中的宏名称本身，这对单行和多行宏都是支持的。%? 将宏名称称为

invoked, whereas `%??` refers to the macro name as *declared*. The two are always the same for case−sensitive macros, but for case−insensitive macros, they can differ.
调用，而% ? ? 指的是声明的宏名称。对于区分大小写的宏，这两者总是相同的，但对于区分大小写的宏，它们可能会有所不同。

For example:
例如:

```
%idefine Foo mov %?,%??
% ,% ?

        foo
        译注:
        FOO
        FOO
```

will expand to:
将扩展至:

```
        mov foo,Foo
        Mov Foo, Foo
        mov FOO,Foo
        Mov FOO, FOO
```

The sequence:
序列:

```
%idefine keyword $%?
定义关键字% ?
```

can be used to make a keyword "disappear", for example in case a new instruction has been used as a label in older code. For example:
可以用来使一个关键字“消失”，例如，如果一个新的指令被用作旧代码中的标签。例如:

```
%idefine pause $%?                  ; Hide the PAUSE instruction
% 指定暂停 $% ? ; 隐藏 PAUSE 指令
```

4.1.6 Undefining Single−Line Macros: `%undef`

4.1.6 Undefining Single-Line Macros:% unf

Single−line macros can be removed with the `%undef` directive. For example, the following sequence:
可以使用% undef 指令删除单行宏。例如，下面的序列:

```
%define foo bar
% 定义 foo bar
%undef  foo
% undef foo

        mov     eax, foo
        Mov eax, foo
```

will expand to the instruction `mov eax, foo`, since after `%undef` the macro `foo` is no longer defined.
将扩展到指令 mov eax，foo，因为在% undef 之后宏 foo 不再定义。

Macros that would otherwise be pre−defined can be undefined on the command−line using the ‘−u’ option on the NASM command line: see section 2.1.20.
可以在命令行上使用 NASM 命令行上的“-u”选项不定义预定义的宏: 参见第 2.1.20 节。

4.1.7 Preprocessor Variables: `%assign`

4.1.7 预处理器变量:% 赋值

An alternative way to define single−line macros is by means of the `%assign` command (and its case−insensitive counterpart `%iassign`, which differs from `%assign` in exactly the same way that `%idefine` differs from `%define`).
定义单行宏的另一种方法是使用% assign 命令(及其不区分大小写的对应命令% iassign，它与% assign 的区别与% idedefine 与% define 的区别完全相同)。

`%assign` is used to define single−line macros which take no parameters and have a numeric value. This value can be specified in the form of an expression, and it will be evaluated once, when the `%assign` directive is processed.
% assign 用于定义没有参数且具有数值的单行宏。这个值可以以表达式的形式指定，当处理% assign 指令时，它将被计算一次。

Like `%define`, macros defined using `%assign` can be re−defined later, so you can do things like
就像% define 一样，使用% assign 定义的宏可以在以后重新定义，所以你可以这样做

```
%assign i i+1
百分比分配 i i + 1
```

to increment the numeric value of a macro.
增加宏的数值。

`%assign` is useful for controlling the termination of `%rep` preprocessor loops: see section 4.5 for an example of this. Another use for `%assign` is given in section 8.4 and section 9.1.
% assign 对于控制% rep 预处理器循环的终止非常有用： 参见第 4.5 节中的示例。% assign 的另一个用法在第 8.4 节和第 9.1 节中给出。

The expression passed to `%assign` is a critical expression (see section 3.8), and must also evaluate to a pure number (rather than a relocatable reference such as a code or data address, or anything involving a register).
传递给% assign 的表达式是一个关键表达式(参见第 3.8 节)，也必须计算为一个纯数(而不是可重定位引用，如代码或数据地址，或任何涉及寄存器的内容)。

4.1.8 Defining Strings: `%defstr`
4.1.8 定义字符串:% defstr

`%defstr`, and its case−insensitive counterpart `%idefstr`, define or redefine a single−line macro without parameters but converts the entire right−hand side, after macro expansion, to a quoted string before definition.
% defstr 及其不区分大小写的对应方% idefstr 定义或重新定义一个没有参数的单行宏，但是在宏展开之后将整个右侧转换为定义之前引用的字符串。

For example:
例如:

```
%defstr test TEST
% defstr 测试 TEST
```

is equivalent to
相当于

```
%define test ’TEST’
% 定义测试“ TEST”
```

This can be used, for example, with the `%!` construct (see section 4.10.2):
例如，可以使用%！构造(参见第 4.10.2 节) :

```
%defstr PATH %!PATH          ; The operating system PATH variable
操作系统 PATH 变量
```

4.1.9 Defining Tokens: `%deftok`
4.1.9 定义令牌:% deftok

`%deftok`, and its case−insensitive counterpart `%ideftok`, define or redefine a single−line macro without parameters but converts the second parameter, after string conversion, to a sequence of tokens.
% deftok 及其不区分大小写的对应方% ideftok 定义或重新定义一个没有参数的单行宏，但在字符串转换后将第二个参数转换为令牌序列。

For example:
例如:

```
%deftok test ’TEST’
% deftok 测试’TEST’
```

is equivalent to
相当于

```
%define test TEST
% 定义测试 TEST
```

4.2 String Manipulation in Macros
4.2 宏中的字符串操作

It’s often useful to be able to handle strings in macros. NASM supports a few simple string handling macro operators from which more complex operations can be constructed.
能够处理宏中的字符串通常很有用。NASM 支持一些简单的字符串处理宏操作符，从中可以构造更复杂的操作。

All the string operators define or redefine a value (either a string or a numeric value) to a single−line macro. When producing a string value, it may change the style of quoting of the input string or strings, and possibly use `\`–escapes inside `‘`–quoted strings.
所有的字符串操作符定义或者重新定义一个值(一个字符串或者一个数值)到一个单行的宏。当生成一个字符串值时，它可能改变输入字符串的引用样式，并且可能在’-quoted 字符串中使用-escape。

4.2.1 Concatenating Strings: `%strcat`
4.2.1 连接字符串:% strcat

The `%strcat` operator concatenates quoted strings and assign them to a single−line macro.
操作符% strcat 连接带引号的字符串并将它们分配给一个单行宏。

For example:
例如:

```
%strcat alpha "Alpha: ", '12" screen'
% strcat Alpha“ Alpha:”, “12”屏幕
```

... would assign the value `'Alpha: 12" screen'` to `alpha`. Similarly:
... 会将值“ Alpha: 12”屏幕”赋给 Alpha:

```
%strcat beta '"foo"\', "'bar'"
% strcat beta“ foo”“ bar”
```

... would assign the value `` `"foo"\\'bar'` `` to `beta`.
将把“ foo”条的值赋给“ beta”。

The use of commas to separate strings is permitted but optional.
使用逗号来分隔字符串是允许的，但是是可选的。

4.2.2 String Length: `%strlen`
4.2.2 String Length:% strlen

The `%strlen` operator assigns the length of a string to a macro. For example:
Strlen 操作符将字符串的长度分配给宏。例如:

```
%strlen charcnt 'my string'
% strlen charcnt'my string
```

In this example, `charcnt` would receive the value 9, just as if an `%assign` had been used. In this example, `'my string'` was a literal string but it could also have been a single−line macro that expands to a string, as in the following example:
在这个例子中，charcnt 将接收值 9，就像使用了% 赋值一样。在这个示例中，‘ my string’是一个字符串，但它也可能是一个单行宏，可以扩展为字符串，如下面的示例所示:

```
%define sometext 'my string'
% define sometext` my string'
%strlen charcnt sometext
一些文字
```

As in the first case, this would result in `charcnt` being assigned the value of 9.
和第一种情况一样，这会导致 charcnt 被赋值为 9。

4.2.3 Extracting Substrings: `%substr`
4.2.3 提取子串:% substr

Individual letters or substrings in strings can be extracted using the `%substr` operator. An example of its use is probably more useful than the description:
字符串中的单个字母或子字符串可以使用% substr 操作符提取。一个使用它的例子可能比描述更有用:

```
%substr mychar 'xyzw'          ; equivalent to %define mychar 'x'
1 %substr mychar 'xyzw'        等价于% define mychar'x'
2 %substr mychar 'xyzw'        ; equivalent to %define mychar 'y'
3 %substr mychar 'xyzw'        等价于% define mychar'y'
2,2 %substr mychar 'xyzw'      ; equivalent to %define mychar 'z'
2,-1 %substr mychar 'xyzw'     等价于% define mychar'z'
2,-2                           ; equivalent to %define mychar 'yz'
% substr mychar'xyzw'1%        等效于% define mychar'yz'
substr mychar'xyzw'2%          ; equivalent to %define mychar 'yzw'
substr mychar'xyzw'3%          等价于% define mychar'yzw'
substr mychar'xyzw'2,2%        ; equivalent to %define mychar 'yz'
substr mychar'xyzw'2,-1%       等效于% define mychar'yz'
substr mychar'xyzw'2,-2
```

As with `%strlen` (see section 4.2.2), the first parameter is the single−line macro to be created and the second is the string. The third parameter specifies the first character to be selected, and the optional fourth parameter preceeded by comma) is the length. Note that the first index is 1, not 0 and the last index is equal to the value that `%strlen` would assign given the same string. Index values out of range result in an empty string. A negative length means "until N−1 characters before the end of string", i.e. `-1` means until end of string, `-2` until one character before, etc.
与% strlen 一样(参见第 4.2.2 节)，第一个参数是要创建的单行宏，第二个参数是字符串。第三个参数指定要选择的第一个字符，第四个可选的参数是长度。注意，第一个索引是 1，而不是 0，最后一个索引等于% strlen 赋给相同字符串的值。Index 值超出范围将导致一个空字符串。负长度表示“ until n-1 characters before the end of string”，即 -1 表示 until end of string,-2 until one character before，等等。

4.3 Multi−Line Macros: `%macro`
4.3 多行宏:% 宏

Multi−line macros are much more like the type of macro seen in MASM and TASM: a multi−line macro definition in NASM looks something like this.
多行宏更像 MASM 和 TASM 中的宏类型: NASM 中的多行宏定义看起来是这样的。

```
%macro  prologue 1
% 宏观序言 1

        push    ebp
        推动 ebp
        mov     ebp,esp
        (尤指)
        sub     esp,%1
        亚特别是% 1

%endmacro
```

```
% endmacro
```

This defines a C−like function prologue as a macro: so you would invoke the macro with a call such as
这将一个类似 c 的函数序言定义为一个宏: 所以你可以通过调用宏来调用宏，比如

```
myfunc:   prologue 12
Myfunc: 序言 12
```

which would expand to the three lines of code
它将扩展到三行代码

```
myfunc: push    ebp
按 ebp
        mov     ebp,esp
        (尤指)
        sub     esp,12
        尤其是，第 12 节
```

The number `1` after the macro name in the `%macro` line defines the number of parameters the macro `prologue` expects to receive. The use of `%1` inside the macro definition refers to the first parameter to the macro call. With a macro taking more than one parameter, subsequent parameters would be referred to as `%2`, `%3` and so on.
宏行% 宏名后面的数字 1 定义了宏序言期望接收的参数数量。宏定义中% 1 的使用指的是宏调用的第一个参数。当一个宏接受多个参数时，后续的参数将被称为% 2、% 3 等。

Multi−line macros, like single−line macros, are case−sensitive, unless you define them using the alternative directive `%imacro`.
多行宏，如单行宏，是区分大小写的，除非您使用替代指令% imacros 来定义它们。

If you need to pass a comma as *part* of a parameter to a multi−line macro, you can do that by enclosing the entire parameter in braces. So you could code things like
如果需要将逗号作为参数的一部分传递给多行宏，可以将整个参数括在大括号中。所以你可以编写如下代码

```
%macro  silly 2
% 宏愚蠢 2

    %2: db
    % 2: 数据      %1
    库             % 1
```

```
%endmacro
% endmacro
        silly 'a', letter_a                 ; letter_a:    db 'a'
        傻乎乎的 a'，字母 _ a                 字母 a:        Db'a'
                                                          db
                                                          数
        silly 'ab', string_ab                 string_ab:   据  'ab'
        愚蠢的 ab，字符串 ab                 ;  字符串 _ ab: 库  " ab"
                                                          db
                                                          数
        silly {13,10}, crlf                   crlf:        据  13,10
        傻瓜{13,10} crlf                    ;  Crlf:        库  13,10
```

4.3.1 Overloading Multi−Line Macros
4.3.1 重载多行宏

As with single−line macros, multi−line macros can be overloaded by defining the same macro name several times with different numbers of parameters. This time, no exception is made for macros with no parameters at all. So you could define
和单行宏一样，多行宏可以通过多次定义相同的宏名称和不同数量的参数来重载。这一次，对于没有参数的宏没有例外。所以你可以定义

```
%macro  prologue 0
% 宏序幕 0

        push    ebp
        推动 ebp
        mov     ebp,esp
        (尤指)

%endmacro
% endmacro
```

to define an alternative form of the function prologue which allocates no local stack space.
定义不分配本地堆栈空间的函数序言的替代形式。

Sometimes, however, you might want to 'overload' a machine instruction; for example, you might want to define
然而，有时你可能想要"重载"一个机器指令; 例如，你可能想要定义一个

```
%macro  push 2
% 宏操作 2

        push    %1
        按% 1
        push    %2
        按% 2

%endmacro
% endmacro
```

so that you could code
这样你就可以编程了

```
        push    ebx             ; this line is not a macro call
        这一行不是宏调用
        push    eax,ecx         ; but this one is
        按 eax，ecx，但这个是
```

Ordinarily, NASM will give a warning for the first of the above two lines, since `push` is now defined to be a macro, and is being invoked with a number of parameters for which no definition has been given. The correct code will still be generated, but the assembler will give a warning. This warning can be disabled by the use of the `-w-macro-params` command−line option (see section 2.1.25).
通常，NASM 会对上面两行中的第一行发出警告，因为 push 现在被定义为一个宏，并且正在使用一些没有给出定义的参数进行调用。正确的代码仍然会被生成，但是汇编器会给出一个警告。这个警告可以通过使用-w-macro-params 命令行选项来禁用(参见 2.1.25 节)。

4.3.2 Macro−Local Labels
4.3.2 宏-本地标签

NASM allows you to define labels within a multi−line macro definition in such a way as to make them local to the macro call: so calling the same macro multiple times will use a different label each time. You do this by prefixing `%%` to the label name. So you can invent an instruction which executes a `RET` if the `Z` flag is set by doing this:
NASM 允许您在多行宏定义中定义标签，以便使它们对宏调用本地化: 因此多次调用同一个宏将每次使用不同的标签。你可以在标签名前加上%% 的前缀。所以你可以发明一个指令来执行 RET，如果 z 标志是这样设置的:

```
%macro  retz 0
% 宏 retz 0

        jnz     %%skip
        Jnz% 跳过
        ret
        后悔
    %%skip:
    % 跳过:

%endmacro
% endmacro
```

You can call this macro as many times as you want, and every time you call it NASM will make up a different ‘real’ name to substitute for the label `%%skip`. The names NASM invents are of the form `..@2345.skip`, where the number 2345 changes with every macro call. The `..@` prefix prevents
您可以随意多次调用此宏，每次调用它时，NASM 都会创建一个不同的“实际”名称来替换标签%% skip。NASM 发明的名字是.@ 2345.skip，其中数字 2345 会随着宏调用的变化而变化。这个。.@ 前缀可以防止

macro−local labels from interfering with the local label mechanism, as described in section 3.9. You should avoid defining your own labels in this form (the `..@` prefix, then a number, then another period) in case they interfere with macro−local labels.
宏本地标签免于干扰本地标签机制，如第 3.9 节所述。你应该避免以这种形式定义你自己的标签。.@ 前缀，然后是一个数字，然后是另一个句号)，以防它们干扰宏本地标签。

4.3.3 Greedy Macro Parameters
4.3.3 贪婪宏参数

Occasionally it is useful to define a macro which lumps its entire command line into one parameter definition, possibly after extracting one or two smaller parameters from the front. An example might be a macro to write a text string to a file in MS−DOS, where you might want to be able to write
有时定义一个宏是有用的，它将整个命令行集中到一个参数定义中，可能是在从前面提取一个或两个较小的参数之后。一个例子可能是一个宏来写一个文本字符串到一个文件在 MS-DOS，你可能希望能够写

```
        writefile [filehandle],"hello, world",13,10
        Writefile [ filehandle ] ，“ hello，world”，13,10
```

NASM allows you to define the last parameter of a macro to be *greedy*, meaning that if you invoke the macro with more parameters than it expects, all the spare parameters get lumped into the last defined one along with the separating commas. So if you code:
NASM 允许您定义一个宏的最后一个参数是贪婪的，这意味着如果您使用比预期更多的参数来调用该宏，那么所有的备用参数将与分隔逗号一起集中到最后一个定义的参数中。因此，如果你编写代码:

```
%macro  writefile 2+
% 宏写文件 2 +

          jmp     %%endstr
          Jmp     %% endstr
   %%str:         db        %2
  %% str:         数据库    % 2
 %%endstr:
%% endstr:
          mov     dx,%%str
       动起来     Dx,%% str
          mov     cx,%%endstr−%%str
       动起来     %% endstr -% str
          mov     bx,%1
       动起来     Bx,% 1
                  ah,0x40
          mov     啊，0 *
       动起来     40
                  0x21
          int     第二季，
         内景     第 21 集

%endmacro
% endmacro
```

then the example call to `writefile` above will work as expected: the text before the first comma, `[filehandle]`, is used as the first macro parameter and expanded when `%1` is referred to, and all the subsequent text is lumped into `%2` and placed after the `db`.
那么上面对 writefile 的示例调用将按预期工作: 第一个逗号[filehandle]之前的文本被用作第一个宏参数，并在引用% 1 时展开，所有后续文本集中到% 2 中并放在 db 之后。

The greedy nature of the macro is indicated to NASM by the use of the `+` sign after the parameter count on the `%macro` line.

宏的贪婪本质通过在% 宏行的参数计数后使用 + 符号向 NASM 表示。

If you define a greedy macro, you are effectively telling NASM how it should expand the macro given *any* number of parameters from the actual number specified up to infinity; in this case, for example, NASM now knows what to do when it sees a call to `writefile` with 2, 3, 4 or more parameters. NASM will take this into account when overloading macros, and will not allow you to define another form of `writefile` taking 4 parameters (for example).
如果您定义了一个贪婪的宏，那么您实际上是在告诉 NASM 如何在给定任意数量的参数(从指定的实际数量到无穷大)的情况下扩展该宏; 例如，在这种情况下，NASM 现在知道当它看到对 writefile 的调用具有 2 个、3 个、4 个或更多的参数时该怎么做。NASM 在重载宏时会考虑到这一点，并且不允许你定义另一种形式的 4 个参数的 writefile (例如)。

Of course, the above macro could have been implemented as a non−greedy macro, in which case the call to it would have had to look like
当然，上面的宏可以作为一个非贪婪宏来实现，在这种情况下，对它的调用必须看起来像

```
      writefile [filehandle], {"hello, world",13,10}
    Writefile [ filehandle ] ,{" hello，world”，13,10}
```

NASM provides both mechanisms for putting commas in macro parameters, and you choose which one you prefer for each macro definition.
NASM 提供了在宏参数中放置逗号的两种机制，你可以为每个宏定义选择一种。

See section 6.3.1 for a better way to write the above macro.
有关编写上述宏的更好方法，请参阅第 6.3.1 节。

4.3.4 Macro Parameters Range
4.3.4 宏参数范围

NASM allows you to expand parameters via special construction `%{x:y}` where `x` is the first parameter index and `y` is the last. Any index can be either negative or positive but must never be zero.
NASM 允许您通过特殊构造% { x: y }展开参数，其中 x 是第一个参数索引，y 是最后一个。任何索引都可以是负的或正的，但绝不能是零。

For example
例如

```
%macro mpar 1-*
% macro mpar 1-*
     db %{3:5}
     Db% {3:5}
```

```
%endmacro
% endmacro

mpar 1,2,3,4,5,6
等号 1,2,3,4,5,6
```

expands to `3,4,5` range.
扩展到 3,4,5 范围。

Even more, the parameters can be reversed so that
更重要的是，这些参数可以被颠倒

```
%macro mpar 1-*
% macro mpar 1-*
     db %{5:3}
     Db% {5:3}
%endmacro
% endmacro

mpar 1,2,3,4,5,6
等号 1,2,3,4,5,6
```

expands to `5,4,3` range.
扩展到 5,4,3 范围。

But even this is not the last. The parameters can be addressed via negative indices so NASM will count them reversed. The ones who know Python may see the analogue here.
但这还不是最后一次。参数可以通过负指数表示，因此 NASM 会将它们反过来计算。了解 Python 的人可能会在这里看到类似的东西。

```
%macro mpar 1-*
% macro mpar 1-*
     db %{-1:-3}
     Db% {-1:-3}
%endmacro
% endmacro

mpar 1,2,3,4,5,6
等号 1,2,3,4,5,6
```

expands to `6,5,4` range.
扩展到 6,5,4 的范围。

Note that NASM uses comma to separate parameters being expanded.
注意，NASM 使用逗号分隔正在展开的参数。

By the way, here is a trick – you might use the index `%{-1:-1}` which gives you the last argument passed to a macro.
顺便说一下，这里有一个技巧-您可以使用索引% {-1:-1}，它给您传递给宏的最后一个参数。

4.3.5 Default Macro Parameters
4.3.5 默认宏参数

NASM also allows you to define a multi−line macro with a *range* of allowable parameter counts. If you do this, you can specify defaults for omitted parameters. So, for example:
NASM 还允许你定义一个允许参数计数范围的多行宏。如果你这样做，你可以为省略的参数指定默认值。例如:

```
%macro  die 0-1 "Painful program death has occurred."
% 宏死亡 0-1“痛苦的程序死亡已经发生。”

        writefile 2,%1
        Writefile 2,% 1
        mov     ax,0x4c01
        Mov ax, 0 x4c01
        int     0x21
        Int 0x21 整体 0x21

%endmacro
% endmacro
```

This macro (which makes use of the `writefile` macro defined in section 4.3.3) can be called with an explicit error message, which it will display on the error output stream before exiting, or it can be called with no parameters, in which case it will use the default error message supplied in the macro definition.
这个宏(使用第 4.3.3 节中定义的 writefile 宏)可以通过显式的错误消息来调用，在退出之前，它将显示在错误输出流中，或者可以在没有参数的情况下调用它，在这种情况下，它将使用宏定义中提供的默认错误消息。

In general, you supply a minimum and maximum number of parameters for a macro of this type; the minimum number of parameters are then required in the macro call, and then you provide defaults for the optional ones. So if a macro definition began with the line
通常，您为这种类型的宏提供最小和最大参数数目; 然后在宏调用中需要最小参数数目，然后为可选参数提供默认值。因此，如果宏定义以行开头

```
%macro foobar 1-3 eax,[ebx+2]
% 宏 foobar 1-3 eax, [ ebx + 2]
```

then it could be called with between one and three parameters, and `%1` would always be taken from the macro call. `%2`, if not specified by the macro call, would default to `eax`, and `%3` if not specified would default to `[ebx+2]`.
然后可以使用一个到三个参数调用它，并且始终从宏调用中获取% 1。如果宏调用没有指定% 2，则默认为 eax，如果没有指定% 3，则默认为[ebx + 2]。

You can provide extra information to a macro by providing too many default parameters:
你可以通过提供太多的默认参数来为宏提供额外的信息:

```
%macro quux 1 something
% macro quux 1 something
```

This will trigger a warning by default; see section 2.1.25 for more information. When `quux` is invoked, it receives not one but two parameters. `something` can be referred to as `%2`. The difference between passing `something` this way and writing `something` in the macro body is that with this way `something` is evaluated when the macro is defined, not when it is expanded.
默认情况下，这将触发一个警告; 有关更多信息，请参见第 2.1.25 节。当 quux 被调用时，它接收到的不是一个而是两个参数。可以称为% 2。以这种方式传递某些内容与在宏主体中写入某些内容之间的区别在于，使用这种方式在定义宏时计算某些内容，而不是在展开宏时计算。

You may omit parameter defaults from the macro definition, in which case the parameter default is taken to be blank. This can be useful for macros which can take a variable number of parameters, since the `%0` token (see section 4.3.6) allows you to determine how many parameters were really passed to the macro call.
你可以从宏定义中省略参数默认值，在这种情况下，参数默认值是空的。这对于可以接受可变参数数量的宏很有用，因为% 0 令牌(参见第 4.3.6 节)允许您确定实际传递给宏调用的参数数量。

This defaulting mechanism can be combined with the greedy-parameter mechanism; so the `die` macro above could be made more powerful, and more useful, by changing the first line of the definition to
这种默认机制可以与贪婪参数机制相结合，因此通过将定义的第一行更改为

```
%macro die 0-1+ "Painful program death has occurred.",13,10
% 宏骰子 0-1 + "痛苦的程序死亡已经发生。"
```

The maximum parameter count can be infinite, denoted by `*`. In this case, of course, it is impossible to provide a *full* set of default parameters. Examples of this usage are shown in section 4.3.8.
最大参数计数可以是无限的，由 * 表示。当然，在这种情况下，不可能提供完整的默认参数集。这种用法的例子见第 4.3.8 节。

4.3.6 `%0`: Macro Parameter Counter
4.3.6% 0: 宏参数计数器

The parameter reference `%0` will return a numeric constant giving the number of parameters received, that is, if `%0` is n then `%n` is the last parameter. `%0` is mostly useful for macros that can take a variable number of parameters. It can be used as an argument to `%rep` (see section 4.5) in order to iterate through all the parameters of a macro. Examples are given in section 4.3.8.
参数引用% 0 将返回一个数值常数，该常数给出接收到的参数数量，也就是说，如果% 0 是 n，那么% n 是最后一个参数。% 0 对于可以接受不同数量参数的宏来说非常有用。它可以作为% rep 的参数(参见第 4.5 节)来迭代一个宏的所有参数。例子在第 4.3.8 节给出。

4.3.7 `%00`: Label Preceeding Macro
4.3.7% 00: Label before Macro

`%00` will return the label preceeding the macro invocation, if any. The label must be on the same line as the macro invocation, may be a local label (see section 3.9), and need not end in a colon.
% 00 将返回宏调用前的标签(如果有的话)。标签必须与宏调用位于同一行，可能是本地标签(参见 3.9 节)，并且不需要以冒号结束。

4.3.8 `%rotate`: Rotating Macro Parameters
4.3.8% 旋转: 旋转宏参数

Unix shell programmers will be familiar with the `shift` shell command, which allows the arguments passed to a shell script (referenced as `$1`, `$2` and so on) to be moved left by one place, so that the argument previously referenced as `$2` becomes available as `$1`, and the argument previously referenced as `$1` is no longer available at all.
Unix shell 程序员将熟悉 shift shell 命令，该命令允许将传递给 shell 脚本(引用为 $1、 $2 等)的参数左移一位，这样以前引用为 $2 的参数就可以作为 $1 使用，而以前引用为 $1 的参数则不再可用。

NASM provides a similar mechanism, in the form of `%rotate`. As its name suggests, it differs from the Unix `shift` in that no parameters are lost: parameters rotated off the left end of the argument list reappear on the right, and vice versa.

NASM 提供了一个类似的机制，以% rotate 的形式。顾名思义，它不同于 Unix shift，因为没有丢失任何参数: 从参数列表左端旋转的参数重新出现在右端，反之亦然。

`%rotate` is invoked with a single numeric argument (which may be an expression). The macro parameters are rotated to the left by that many places. If the argument to `%rotate` is negative, the macro parameters are rotated to the right.
使用单个数值参数(可能是一个表达式)调用`% rotate`。宏参数向左旋转了这么多位置。如果`% rotate` 的参数是负的，宏参数就会向右旋转。

So a pair of macros to save and restore a set of registers might work as follows:
因此，保存和恢复一组寄存器的一对宏可能工作如下:

```
%macro  multipush 1-*
% 宏多次推送 1-*

  %rep  %0
  % rep% 0
        push      %1
        推        % 1
  %rotate 1
  百分比旋转 1
  %endrep
  百分比

%endmacro
% endmacro
```

This macro invokes the `PUSH` instruction on each of its arguments in turn, from left to right. It begins by pushing its first argument, `%1`, then invokes `%rotate` to move all the arguments one place to the left, so that the original second argument is now available as `%1`. Repeating this procedure as many times as
这个宏依次从左到右调用每个参数的 PUSH 指令。它首先推送第一个参数% 1，然后调用% rotate 将所有参数向左移动一位，这样原始的第二个参数现在就可以作为% 1 使用了。重复这个过程

there were arguments (achieved by supplying `%0` as the argument to `%rep`) causes each argument in turn to be pushed.
有一些参数(通过将% 0 作为参数提供给% rep 来实现)导致每个参数依次被推送。

Note also the use of `*` as the maximum parameter count, indicating that there is no upper limit on the number of parameters you may supply to the `multipush` macro.
还要注意使用 * 作为最大参数计数，表明可以向多推宏提供的参数数量没有上限。

It would be convenient, when using this macro, to have a `POP` equivalent, which *didn't* require the arguments to be given in reverse order. Ideally, you would write the `multipush` macro call, then cut−and−paste the line to where the pop needed to be done, and change the name of the called macro to `multipop`, and the macro would take care of popping the registers in the opposite order from the one in which they were pushed.
当使用这个宏的时候，有一个 POP 等价物是很方便的，这样就不需要按照相反的顺序给出参数。理想情况下，您可以编写 multipush 宏调用，然后剪切并粘贴该行到需要执行 pop 的地方，并将调用的宏的名称更改为 multipop，宏将负责按与推送寄存器顺序相反的顺序弹出寄存器。

This can be done by the following definition:
这可以通过下列定义来实现:

```
%macro  multipop 1-*
% 宏 multipop 1-*

  %rep %0
  % rep% 0
  %rotate -1
  % rotate-1 旋转-1
        pop     %1
        Pop% 1
  %endrep
  百分比

%endmacro
% endmacro
```

This macro begins by rotating its arguments one place to the *right*, so that the original *last* argument appears as `%1`. This is then popped, and the arguments are rotated right again, so the second−to−last argument becomes `%1`. Thus the arguments are iterated through in reverse order.
这个宏首先将它的参数向右旋转一个位置，这样最初的最后一个参数显示为% 1。然后弹出，参数再次向右旋转，所以倒数第二个参数变成% 1。因此这些参数是以相反的顺序迭代的。

4.3.9 Concatenating Macro Parameters
4.3.9 连接宏参数

NASM can concatenate macro parameters and macro indirection constructs on to other text surrounding them. This allows you to declare a family of symbols, for example, in a macro definition. If, for example, you wanted to generate a table of key codes along with offsets into the table, you could code something like
NASM 可以将宏参数和宏间接构造连接到它们周围的其他文本上。这允许你声明一系列符号，例如，在宏定义中。例如，如果你想要生成一个关键代码的表格，并且在表格中加入偏移量，你可以编写如下代码

```
%macro keytab_entry 2
% 宏 keytab _ 条目 2

    keypos%1    equ     $-keytab
    Keypos% 1 equ $- keytab
                db      %2
                Db% 2

%endmacro
```

```
% endmacro

keytab:
关键字:
          keytab_entry F1,128+1
          Keytab _ entry F1,128 + 1
          keytab_entry F2,128+2
          Keytab _ entry F2,128 + 2
          keytab_entry Return,13
          Keytab _ entry Return，13 keytab _ entry 返回，13
```

which would expand to
会扩展到

```
keytab:
关键字:
keyposF1        equ     $-keytab
键盘 1          等等    键盘
                db
                数据    128+1
                库      128 + 1
keyposF2        equ     $-keytab
Keyposf2 键盘   等等    键盘
                db
                数据    128+2
                库      128 + 2
keyposReturn    equ     $-keytab
钥匙归还        等等    键盘
                db
                数据
                库      13
```

You can just as easily concatenate text on to the other end of a macro parameter, by writing `%1foo`.
通过编写% 1foo，可以很容易地将文本连接到宏参数的另一端。

If you need to append a *digit* to a macro parameter, for example defining labels `foo1` and `foo2` when passed the parameter `foo`, you can't code `%11` because that would be taken as the eleventh macro parameter. Instead, you must code `%{1}1`, which will separate the first `1` (giving the number of the macro parameter) from the second (literal text to be concatenated to the parameter).
如果需要在宏参数后面追加一个数字，例如在传递参数 foo 时定义标签 foo1 和 foo2，则无法编码% 11，因为这将被视为第 11 个宏参数。相反，您必须编写% {1}1，它将第一个 1(给出宏参数的数目)与第二个(连接到参数的文本)分开。

This concatenation can also be applied to other preprocessor in−line objects, such as macro−local labels (section 4.3.2) and context−local labels (section 4.7.2). In all cases, ambiguities in syntax can be resolved by enclosing everything after the `%` sign and before the literal text in braces: so `%{%foo}bar` concatenates the text `bar` to the end of the real name of the macro−local label `%%foo`. (This is unnecessary, since the form NASM uses for the real names of macro−local labels means that the two usages `%{%foo}bar` and `%%foobar` would both expand to the same thing anyway; nevertheless, the capability is there.)
这种连接还可以应用于其他预处理器行内对象，如宏本地标签(第 4.3.2 节)和上下文本地标签(第 4.7.2 节)。在所有情况下，语法中的歧义都可以通过将所有内容放在% 符号之后和大括号中的文本之前来解决: 因此% {% foo } bar 将文本栏连接到宏-本地标签%% foo 的实际名称的末尾。(这是没有必要的，因为 NASM 用于宏本地标签实名的表单意味着% {% foo } bar 和%% foobar 这两个用法无论如何都会扩展到同一个东西; 然而，功能是存在的。)

The single−line macro indirection construct, `%[...]` (section 4.1.3), behaves the same way as macro parameters for the purpose of concatenation.
单行宏间接构造% [...](第 4.1.3 节)的行为方式与宏参数的行为方式相同，用于连接。

See also the `%+` operator, section 4.1.4.
参见% + 操作符，第 4.1.4 节。

4.3.10 Condition Codes as Macro Parameters
4.3.10 条件码作为宏参数

NASM can give special treatment to a macro parameter which contains a condition code. For a start, you can refer to the macro parameter `%1` by means of the alternative syntax `%+1`, which informs NASM that this macro parameter is supposed to contain a condition code, and will cause the preprocessor to report an error message if the macro is called with a parameter which is *not* a valid condition code.
NASM 可以对包含条件代码的宏参数进行特殊处理。首先，您可以通过替代语法% + 1 引用宏参数% 1，它通知 NASM 这个宏参数应该包含条件代码，并且如果调用的宏参数不是有效的条件代码，它将导致预处理器报告错误消息。

Far more usefully, though, you can refer to the macro parameter by means of `%-1`, which NASM will expand as the *inverse* condition code. So the `retz` macro defined in section 4.3.2 can be replaced by a general conditional−return macro like this:
但是，更有用的是，您可以通过%-1 引用宏参数，NASM 将其扩展为逆条件代码。所以在第 4.3.2 节中定义的 retz 宏可以被一个通用的条件返回宏代替，如下所示:

```
%macro  retc 1
% 宏 retc 1

        j%-1    %%skip
        J%-1% 跳过
        ret
        后悔
  %%skip:
  % 跳过:

%endmacro
% endmacro
```

This macro can now be invoked using calls like `retc ne`, which will cause the conditional−jump instruction in the macro expansion to come out as `JE`, or `retc po` which will make the jump a `JPE`.

现在可以使用诸如 retc ne 之类的调用来调用这个宏，这将导致宏展开中的条件跳转指令显示为 JE，或者 retc po，这将使跳转成为 JPE。

The `%+1` macro−parameter reference is quite happy to interpret the arguments `CXZ` and `ECXZ` as valid condition codes; however, `%-1` will report an error if passed either of these, because no inverse condition code exists.
% + 1 宏参数引用非常乐意将参数 CXZ 和 ECXZ 解释为有效的条件代码; 但是，如果传递了这两个参数中的任何一个,%-1 将报告一个错误，因为不存在逆条件代码。

4.3.11 Disabling Listing Expansion
4.3.11 禁用上市扩展

When NASM is generating a listing file from your program, it will generally expand multi−line macros by means of writing the macro call and then listing each line of the expansion. This allows you to see which instructions in the macro expansion are generating what code; however, for some macros this clutters the listing up unnecessarily.
当 NASM 从您的程序生成一个列表文件时，它通常通过编写宏调用并列出扩展的每一行来展开多行宏。这可以让你看到宏展开中的哪些指令正在生成哪些代码; 然而，对于某些宏来说，这会不必要地打乱列表。

NASM therefore provides the `.nolist` qualifier, which you can include in a macro definition to inhibit the expansion of the macro in the listing file. The `.nolist` qualifier comes directly after the number of parameters, like this:
因此，NASM 提供了。Nolist 限定符，可以包含在宏定义中，以抑制列表文件中宏的扩展。这个。Nolist 限定符紧跟在参数数量之后，如下所示:

```
%macro foo 1.nolist
% macro foo 1. nolist
```

Or like this:
或者像这样:

```
%macro bar 1−5+.nolist a,b,c,d,e,f,g,h
% 宏条 1-5 + . nolist a，b，c，d，e，f，g，h
```

4.3.12 Undefining Multi−Line Macros: `%unmacro`
4.3.12 Undefining Multi-Line Macros:% unmacro

Multi−line macros can be removed with the `%unmacro` directive. Unlike the `%undef` directive, however, `%unmacro` takes an argument specification, and will only remove exact matches with that argument specification.
可以使用% unmacro 指令删除多行宏。然而，与% undef 指令不同的是,% unmacro 接受一个参数规范，并且只会删除与该参数规范完全匹配的参数。

For example:
例如:

```
%macro foo 1-3
% 宏 foo 1-3
        ; Do something
        做点什么
%endmacro %unmac
ro foo 1-3
% endmacro%
unmacro foo 1-3
```

removes the previously defined macro `foo`, but
删除先前定义的宏 foo，但是

```
%macro bar 1-3
% 宏条 1-3
        ; Do something
        做点什么
%endmacro %unm
acro bar 1
% endmacro%
unmacro bar 1
```

does *not* remove the macro `bar`, since the argument specification does not match exactly.
没有删除宏条，因为参数规范并不完全匹配。

4.4 Conditional Assembly
4.4 条件集合

Similarly to the C preprocessor, NASM allows sections of a source file to be assembled only if certain conditions are met. The general syntax of this feature looks like this:
与 c 预处理器类似，NASM 允许只有在满足特定条件的情况下才能组装源文件的部分。这个特性的一般语法如下:

```
%if<condition>
% (如 < 条件 >)
    ; some code which only appears if <condition> is
met %elif<condition2>
    一些代码，只有在满足% elif < condition 2 > 时才出现
    ; only appears if <condition> is not met but <condition2> is
    只有在不满足 < 条件 > 而 < 条件 2 > 满足 < 条件 2 > 时才出现
%else
其他%
    ; this appears if neither <condition> nor <condition2> was met
    如果既不满足 < 条件 > 也不满足 < 条件 2 > ，则会出现这种情况
%endif
% endif
```

The inverse forms `%ifn` and `%elifn` are also supported.
也支持反式形式% ifn 和% elifn。

The `%else` clause is optional, as is the `%elif` clause. You can have more than one `%elif` clause as well.
Else 子句和% elif 子句都是可选的，你也可以有多个% elif 子句。

There are a number of variants of the `%if` directive. Each has its corresponding `%elif`, `%ifn`, and `%elifn` directives; for example, the equivalents to the `%ifdef` directive are `%elifdef`, `%ifndef`, and `%elifndef`.
If 指令有许多变体。每个指令都有相应的% elif、% ifn 和% elifn 指令; 例如,% ifdef 指令的等价物是% elifdef、% ifndef 和% elifdef。

4.4.1 `%ifdef`: Testing Single−Line Macro Existence
4.4.1% ifdef: 测试单行宏存在

Beginning a conditional−assembly block with the line `%ifdef MACRO` will assemble the subsequent code if, and only if, a single−line macro called `MACRO` is defined. If not, then the `%elif` and `%else` blocks (if any) will be processed instead.
以行% ifdef MACRO 开始一个条件程序集块，当且仅当定义了一个名为 MACRO 的单行宏时，才会组装后续代码。如果没有，那么% elif 和% else 块(如果有的话)将被处理。

For example, when debugging a program, you might want to write code such as
例如，当调试一个程序时，你可能需要编写如下代码

```
          ; perform some function
          发挥一些作用
%ifdef DEBUG
% ifdef DEBUG
          writefile 2,"Function performed successfully",13,10
          Writefile 2，“ Function performed successfully”，13,10
%endif
% endif
          ; go and do something else
          去做点别的事吧
```

Then you could use the command−line option `−dDEBUG` to create a version of the program which produced debugging messages, and remove the option to generate the final release version of the program.
然后，可以使用命令行选项 dDEBUG 创建产生调试消息的程序版本，并删除生成程序最终发布版本的选项。

You can test for a macro *not* being defined by using `%ifndef` instead of `%ifdef`. You can also test for macro definitions in `%elif` blocks by using `%elifdef` and `%elifndef`.
您可以使用% ifndef 而不是% ifdef 来测试未定义的宏。你也可以使用% elif 和% elifdef 来测试% elif 块中的宏定义。

4.4.2 `%ifmacro`: Testing Multi−Line Macro Existence
4.4.2% ifmacro: 测试多行宏的存在

The `%ifmacro` directive operates in the same way as the `%ifdef` directive, except that it checks for the existence of a multi−line macro.
Ifmacro 指令与 ifdef 指令的操作方式相同，只是它检查是否存在多行宏。

For example, you may be working with a large project and not have control over the macros in a library. You may want to create a macro with one name if it doesn't already exist, and another name if one with that name does exist.
例如，您可能正在处理一个大型项目，而无法控制库中的宏。如果宏还不存在，你可能想创建一个有一个名字的宏，如果有另一个名字的宏存在，你可能想创建另一个名字的宏。

The `%ifmacro` is considered true if defining a macro with the given name and number of arguments would cause a definitions conflict. For example:
如果用给定的名称和参数数量定义一个宏会导致定义冲突，那么% ifmacro 被认为是真的。例如:

```
%ifmacro MyMacro 1-3
% 如果宏 MyMacro 1-3

     %error "MyMacro 1-3" causes a conflict with an existing macro.
     % 错误" MyMacro 1-3"与现有宏发生冲突。

%else
其他%

     %macro MyMacro 1-3
     % 宏 MyMacro 1-3

             ; insert code to define the macro
             插入代码来定义宏

     %endmacro
     % endmacro

%endif
% endif
```

This will create the macro "MyMacro 1−3" if no macro already exists which would conflict with it, and emits a warning if there would be a definition conflict.
这将创建宏“ MyMacro 1-3”，如果没有宏已经存在，这将与它冲突，并发出警告，如果有一个定义冲突。

You can test for the macro not existing by using the `%ifnmacro` instead of `%ifmacro`. Additional tests can be performed in `%elif` blocks by using `%elifmacro` and `%elifnmacro`.
可以使用% ifnmacro 而不是% ifmacro 来测试不存在的宏。额外的测试可以使用% elifmacro 和% elifmacro 在% elif 块中执行。

4.4.3 `%ifctx`: Testing the Context Stack
4.4.3% ifctx: 测试 Context Stack

The conditional−assembly construct `%ifctx` will cause the subsequent code to be assembled if and only if the top context on the preprocessor's context stack has the same name as one of the arguments. As with `%ifdef`, the inverse and `%elif` forms `%ifnctx`, `%elifctx` and `%elifnctx` are also supported.

条件汇编构造% ifctx 将导致后续代码的汇编，当且仅当预处理器上下文堆栈的顶部上下文与其中一个参数的名称相同时。与% ifdef 一样，也支持反向和% elif 形式% ifnctx、% elifctx 和% elifctx。

For more details of the context stack, see section 4.7. For a sample use of `%ifctx`, see section 4.7.6.
有关上下文堆栈的更多细节，请参见第 4.7 节。有关% ifctx 的示例使用，请参见第 4.7.6 节。

4.4.4 `%if`: Testing Arbitrary Numeric Expressions
4.4.4% if: Testing Arbitrary Numeric Expressions

The conditional−assembly construct `%if expr` will cause the subsequent code to be assembled if and only if the value of the numeric expression `expr` is non−zero. An example of the use of this feature is in deciding when to break out of a `%rep` preprocessor loop: see section 4.5 for a detailed example.
条件程序集构造% if expr 将导致后续代码的组装，当且仅当数值表达式 expr 的值非零时。使用此特性的一个例子是决定何时中断% rep 预处理器循环: 请参阅第 4.5 节了解详细示例。

The expression given to `%if`, and its counterpart `%elif`, is a critical expression (see section 3.8).
给予% if 及其对应的% elif 的表达式是一个关键表达式(参见第 3.8 节)。

`%if` extends the normal NASM expression syntax, by providing a set of relational operators which are not normally available in expressions. The operators `=`, `<`, `>`, `<=`, `>=` and `<>` test equality, less−than, greater−than, less−or−equal, greater−or−equal and not−equal respectively. The C−like forms `==` and `!=` are supported as alternative forms of = and <>. In addition, low−priority logical operators `&&`, `^^` and
% if 通过提供一组通常在表达式中不可用的关系运算符来扩展正常的 NASM 表达式语法。运算符 = 、 < 、 > 、 < = 、 > 和 < > 分别检验等式、小于、大于、小于或等于、大于或等于和不等于。类 c 形式 = = 和! = 被支持为 = 和 < > 的替代形式。此外，低优先级的逻辑运算符 & & ，^ ^ 和
`||` are provided, supplying logical AND, logical XOR and logical OR. These work like the C logical operators (although C has no logical XOR), in that they always return either 0 or 1, and treat any non−zero input as 1 (so that `^^`, for example, returns 1 if exactly one of its inputs is zero, and 0 otherwise). The relational operators also return 1 for true and 0 for false.
提供了逻辑和、逻辑异或和逻辑或。它们的工作方式类似于 c 逻辑运算符(尽管 c 没有逻辑 XOR)，因为它们总是返回 0 或 1，并将任何非零输入视为 1(例如 ^ ^ ，如果其输入之一为零，则返回 1，否则返回 0)。关系运算符也返回 1 表示真，0 表示假。

56

Like other `%if` constructs, `%if` has a counterpart `%elif`, and negative forms `%ifn` and `%elifn`.
像其他% if 构造一样,% if 有一个对应的% elif，否定形式% ifn 和% elif。

4.4.5 `%ifidn` and `%ifidni`: Testing Exact Text Identity
4.4.5% ifidn 和% ifidni: 测试确切的文本身份

The construct `%ifidn text1,text2` will cause the subsequent code to be assembled if and only if `text1` and `text2`, after expanding single−line macros, are identical pieces of text. Differences in white space are not counted.
构造% ifidn text1，text2 将导致后续代码的组装，当且仅当 text1 和 text2 在展开单行宏之后是相同的文本片段时。不计算空白区域的差异。

`%ifidni` is similar to `%ifidn`, but is case−insensitive.
% ifidni 与% ifidn 相似，但不区分大小写。

For example, the following macro pushes a register or number on the stack, and allows you to treat `IP` as a real register:
例如，下面的宏将一个寄存器或数字推送到堆栈上，并允许您将 IP 视为一个真正的寄存器:

```
%macro  pushparam 1
% macro pushparam 1

  %ifidni %1,ip
  % ifidni% 1，ip
        call    %%label
        呼叫%% 标签
  %%label:
  % 标签:
  %else
  其他%
        push    %1
        按% 1
  %endif
  % endif

%endmacro
% endmacro
```

Like other `%if` constructs, `%ifidn` has a counterpart `%elifidn`, and negative forms `%ifnidn` and `%elifnidn`. Similarly, `%ifidni` has counterparts `%elifidni`, `%ifnidni` and `%elifnidni`.
与其他% 如果构造一样,% ifidn 具有对应的% elifidn，以及负形式% ifnidn 和% elifnidn。同样,% ifidni 有对应的% elifidni,% ifnidni 和% elifnidni。

4.4.6 `%ifid`, `%ifnum`, `%ifstr`: Testing Token Types
4.4.6% ifid,% ifnum,% ifstr: Testing Token Types

Some macros will want to perform different tasks depending on whether they are passed a number, a string, or an identifier. For example, a string output macro might want to be able to cope with being passed either a string constant or a pointer to an existing string.
一些宏想要执行不同的任务，这取决于它们传递的是一个数字、一个字符串还是一个标识符。例如，一个字符串输出宏可能希望能够处理传递一个字符串常量或一个指向现有字符串的指针。

The conditional assembly construct `%ifid`, taking one parameter (which may be blank), assembles the subsequent code if and only if the first token in the parameter exists and is an identifier. `%ifnum` works similarly, but tests for the token being a numeric constant; `%ifstr` tests for it being a string.
条件程序集构造% ifid 接受一个参数(可能是空的)，当且仅当参数中的第一个标记存在并且是标识符时才组装后续代码。% ifnum 的工作原理类似，但测试令牌是否为数字常量;% ifstr 测试令牌是否为字符串。

For example, the `writefile` macro defined in section 4.3.3 can be extended to take advantage of `%ifstr` in the following fashion:
例如，在第 4.3.3 节中定义的 writefile 宏可以通过以下方式扩展来利用% ifstr:

```
%macro writefile 2-3+
% 宏写文件 2-3 +

  %ifstr %2
  % ifstr% 2
        jmp         %%endstr
        Jmp         %% endstr
    %if %0 = 3
    % 如果% 0 = 3
      %%str:        db          %2,%3
      %% str:       数据库      % 2,% 3
    %else
    其他%
      %%str:        db          %2
      %% str:       数据库      % 2
    %endif
    % endif
      %%endstr: mov             dx,%%str
      %% endstr: mov            Dx,%% str
                    mov         cx,%%endstr−%%str
                    动起来      %% endstr -% str
  %else
  其他%
                    mov         dx,%2
                    动起来      Dx,% 2
                    mov         cx,%3
                    动起来      Cx,% 3
  %endif
  % endif
                    mov         bx,%1
                    动起来      Bx,% 1
                    mov         ah,0x40
                    动起来      啊，0 * 40
                    int         0x21
                    内景        第二季，第 21 集
```

```
%endmacro
% endmacro
```

Then the `writefile` macro can cope with being called in either of the following two ways:
然后 writefile 宏可以通过以下两种方式来处理调用:

```
        writefile [file], strpointer, length
        Writefile [ file ] , strpointer, length
        writefile [file], "hello", 13, 10
        Writefile [ file ] , " hello", 13,10
```

In the first, `strpointer` is used as the address of an already−declared string, and `length` is used as its length; in the second, a string is given to the macro, which therefore declares it itself and works out the address and length for itself.
在第一种情况下，strpointer 被用作已经声明的字符串的地址，length 被用作它的长度; 在第二种情况下，一个字符串被给予宏，因此宏自己声明它并计算出它自己的地址和长度。

Note the use of `%if` inside the `%ifstr`: this is to detect whether the macro was passed two arguments (so the string would be a single string constant, and `db %2` would be adequate) or more (in which case, all but the first two would be lumped together into `%3`, and `db %2,%3` would be required).
如果在% ifstr 中，请注意% 的用法: 这是为了检测宏是否传递了两个参数(因此字符串将是单个字符串常量，db% 2 就足够了)或更多(在这种情况下，除了前两个参数之外的所有参数都将集中到% 3 中，并且需要 db% 2 和% 3)。

The usual `%elif`..., `%ifn`..., and `%elifn`... versions exist for each of `%ifid`, `%ifnum` and `%ifstr`.
对于% ifid、% ifnum 和% ifstr 的每个版本，都存在通常的% elif... 、% ifn... 和% elif... 版本。

4.4.7 `%iftoken`: Test for a Single Token
4.4.7% iftoken: 测试单个令牌

Some macros will want to do different things depending on if it is passed a single token (e.g. paste it to something else using `%+`) versus a multi−token sequence.
根据传递的是单个令牌(例如，使用% + 将其粘贴到其他东西)还是多个令牌序列，一些宏可能希望执行不同的操作。

The conditional assembly construct `%iftoken` assembles the subsequent code if and only if the expanded parameters consist of exactly one token, possibly surrounded by whitespace.
条件组装构造% iftoken 组装后续代码当且仅当展开的参数正好包含一个标记，可能包含空格。

For example:
例如:

```
%iftoken 1
% iftoken 1
```

will assemble the subsequent code, but
将汇编后续的代码，但

```
%iftoken -1
% iftoken-1
```

will not, since `-1` contains two tokens: the unary minus operator `-`, and the number `1`.
不会，因为 -1 包含两个标记: 一元减去运算符-和数字 1。

The usual `%eliftoken`, `%ifntoken`, and `%elifntoken` variants are also provided.
通常的% eliftoken、% ifntoken 和% eliftoken 变体也被提供。

4.4.8 `%ifempty`: Test for Empty Expansion
4.4.8% ifempty: Test for Empty Expansion

The conditional assembly construct `%ifempty` assembles the subsequent code if and only if the expanded parameters do not contain any tokens at all, whitespace excepted.
条件程序集构造% ifempty 当且仅当展开的参数根本不包含任何标记，空格除外。

The usual `%elifempty`, `%ifnempty`, and `%elifnempty` variants are also provided.
还提供了通常的% elifempty、% ifneempty 和% elifnevempty 变体。

4.4.9 `%ifenv`: Test If Environment Variable Exists
4.4.9% ifenv: 测试环境变量是否存在

The conditional assembly construct `%ifenv` assembles the subsequent code if and only if the environment variable referenced by the `%!`*variable* directive exists.
条件程序集构造% ifenv 组装后续代码，当且仅当%！变量指令存在。

The usual `%elifenv`, `%ifnenv`, and `%elifnenv` variants are also provided.
还提供了通常的% elifenv,% ifnenv 和% elifnenv 变体。

Just as for `%!`*variable* the argument should be written as a string if it contains characters that would not be legal in an identifier. See section 4.10.2.
就像% 一样！变量的参数应该写成字符串，如果它包含的字符在标识符中是不合法的。参见第 4.10.2 节。

4.5 Preprocessor Loops: `%rep`
4.5 预处理器循环:% rep

NASM's `TIMES` prefix, though useful, cannot be used to invoke a multi−line macro multiple times, because it is processed by NASM after macros have already been expanded. Therefore NASM provides another form of loop, this time at the preprocessor level: `%rep`.
NASM 的 TIMES 前缀虽然有用，但不能多次调用多行宏，因为它是在宏已经扩展之后由 NASM 处理的。因此 NASM 提供了另一种形式的循环，这次是在预处理器级别:% rep。

The directives `%rep` and `%endrep` (`%rep` takes a numeric argument, which can be an expression; `%endrep` takes no arguments) can be used to enclose a chunk of code, which is then replicated as many times as specified by the preprocessor:
指令% rep 和% endrep (% rep 接受一个数值参数，可以是一个表达式;% endrep 不接受任何参数) 可以用来封装一段代码，然后根据预处理器的指定复制这段代码的次数:

```
%assign
转让百分
比        i 0
%rep
% rep 百
分比      64
          inc
          股份有      word [table+2*i]
          限公司      Word [表 + 2 * i ]
%assign
转让百分    i+1
比        i  I + 1
%endrep
百分比
```

This will generate a sequence of 64 `INC` instructions, incrementing every word of memory from `[table]` to `[table+126]`.
这将生成一个由 64 个 INC 指令组成的序列，从[table]递增到[table + 126]。

For more complex termination conditions, or to break out of a repeat loop part way along, you can use the `%exitrep` directive to terminate the loop, like this:
对于更复杂的终止条件，或者中断重复循环，可以使用% exitrep 指令终止循环，如下所示:

```
fibonacci:
斐波那契:
%assign i 0
% 赋值 i 0
%assign j 1
百分比分配 j1
%rep 100
% rep 100
%if j > 65535
% ，如果 j > 65535
    %exitrep
    % exitrep 出境
%endif
% endif
        dw j
        Dw j
%assign k j+i
分配 kj + i 的百分比
%assign i j
% 赋值 i j
%assign j k
百分比分配 j k
%endrep
百分比

fib_number equ ($-fibonacci)/2
Fib _ number equ ($- fibonacci)/2
```

This produces a list of all the Fibonacci numbers that will fit in 16 bits. Note that a maximum repeat count must still be given to `%rep`. This is to prevent the possibility of NASM getting into an infinite loop in the preprocessor, which (on multitasking or multi−user systems) would typically cause all the system memory to be gradually used up and other applications to start crashing.

这会生成一个 16 位 Fibonacci 数字的列表。注意，最大重复计数仍然必须给% rep。这是为了防止 NASM 在预处理器中进入无限循环的可能性，这种循环(在多任务或多用户系统上)通常会导致所有系统内存逐渐耗尽，其他应用程序开始崩溃。

Note a maximum repeat count is limited by 62 bit number, though it is hardly possible that you ever need anything bigger.
请注意，最大重复计数是有限的 62 位数字，虽然这是几乎不可能的，你曾经需要任何更大的。

4.6 Source Files and Dependencies
4.6 源文件和依赖项

These commands allow you to split your sources into multiple files.
这些命令允许您将源文件分割为多个文件。

4.6.1 `%include`: Including Other Files
4.6.1% 包括: 包括其他文件

Using, once again, a very similar syntax to the C preprocessor, NASM's preprocessor lets you include other source files into your code. This is done by the use of the `%include` directive:
再次使用与 c 预处理器非常相似的语法，NASM 的预处理器允许您在代码中包含其他源文件。这是通过使用% include 指令来完成的:

```
%include "macros.mac"
% 包括" macros.mac"
```

will include the contents of the file `macros.mac` into the source file containing the `%include` directive.
将把文件 macros.mac 的内容包含到包含% include 指令的源文件中。

Include files are searched for in the current directory (the directory you're in when you run NASM, as opposed to the location of the NASM executable or the location of the source file), plus any directories specified on the NASM command line using the `-i` option.
Include 文件在当前目录(运行 NASM 时所在的目录，而不是 NASM 可执行文件的位置或源文件的位置)以及使用 -i 选项在 NASM 命令行上指定的任何目录中进行搜索。

The standard C idiom for preventing a file being included more than once is just as applicable in NASM:
防止一个文件被包含超过一次的标准 c 语言习惯用法同样适用于 NASM:
if the file `macros.mac` has the form
如果 macros.mac 文件有这个表单

```
%ifndef MACROS_MAC
% ifndef MACROS _ mac
    %define MACROS_MAC
    % 定义 MACROS _ mac
    ; now define some macros
    现在定义一些宏
%endif
% endif
```

then including the file more than once will not cause errors, because the second time the file is included nothing will happen because the macro MACROS_MAC will already be defined.
那么多次包含该文件将不会导致错误，因为第二次包含该文件将不会发生任何事情，因为宏 macros_mac 将已经被定义。

You can force a file to be included even if there is no %include directive that explicitly includes it, by using the -p option on the NASM command line (see section 2.1.18).
通过在 NASM 命令行上使用 -p 选项，即使没有显式包含文件的% include 指令，也可以强制包含文件(参见 2.1.18 节)。

4.6.2 %pathsearch: Search the Include Path
4.6.2% 路径搜索: 搜索包含路径

The %pathsearch directive takes a single−line macro name and a filename, and declare or redefines the specified single−line macro to be the include−path−resolved version of the filename, if the file exists (otherwise, it is passed unchanged.)
% pathsearch 指令接受一个单行宏名和一个文件名，并声明或重新定义指定的单行宏作为文件名的包含路径解析版本(如果文件存在的话)(否则，传递的内容不变)

For example,
例如,

```
%pathsearch MyFoo "foo.bin"
% pathsearch MyFoo“ foo.bin”
```

... with -Ibins/ in the include path may end up defining the macro MyFoo to be "bins/foo.bin".
在包含路径中使用 -Ibins/可能最终会将宏 MyFoo 定义为“ bins/foo.bin”。

4.6.3 %depend: Add Dependent Files
4.6.3% 依赖: 添加依赖文件

The %depend directive takes a filename and adds it to the list of files to be emitted as dependency generation when the -M options and its relatives (see section 2.1.4) are used. It produces no output.
当使用 -m 选项及其相关项(参见 2.1.4 节)时,% depend 指令接受一个文件名，并将其添加到作为依赖项生成发出的文件列表中。它不产生任何输出。

This is generally used in conjunction with %pathsearch. For example, a simplified version of the standard macro wrapper for the INCBIN directive looks like:
这通常与% 路径搜索结合使用。例如，INCBIN 指令的标准宏包装器的简化版本如下:

```
%imacro incbin 1-2+ 0
% imacroincbin 1-2 + 0
%pathsearch dep %1
% 路径搜索深度% 1
%depend dep
% 依赖于
        incbin dep,%2
        2 号病房
%endmacro
% endmacro
```

This first resolves the location of the file into the macro dep, then adds it to the dependency lists, and finally issues the assembler−level INCBIN directive.
这首先将文件的位置解析到宏 dep 中，然后将其添加到依赖项列表中，最后发出汇编器级 INCBIN 指令。

4.6.4 %use: Include Standard Macro Package
4.6.4% 使用: 包含标准宏包

The %use directive is similar to %include, but rather than including the contents of a file, it includes a named standard macro package. The standard macro packages are part of NASM, and are described in chapter 5.

% use 指令与% include 类似，但是它不包括文件的内容，而是包括一个命名的标准宏包。标准宏包是 NASM 的一部分，在第 5 章中有描述。

Unlike the `%include` directive, package names for the `%use` directive do not require quotes, but quotes are permitted. In NASM 2.04 and 2.05 the unquoted form would be macro−expanded; this is no longer true. Thus, the following lines are equivalent:
与% include 指令不同,% use 指令的包名不需要引号，但允许引号。在 NASM 2.04 和 2.05 中，不带引号的表单将被宏扩展; 这不再是真的。因此，下面几行是等价的:

```
%use altreg
% 使用 altreg
%use 'altreg'
% 使用" altreg"
```

Standard macro packages are protected from multiple inclusion. When a standard macro package is used, a testable single−line macro of the form `__USE_`*package*`__` is also defined, see section 4.11.9.
标准的宏包受到多重包含的保护。当使用标准宏包时，还定义了表单 _ _ USE _ package _ _ 的可测试单行宏，参见第 4.11.9 节。

4.7 The Context Stack
4.7 上下文堆栈

Having labels that are local to a macro definition is sometimes not quite powerful enough: sometimes you want to be able to share labels between several macro calls. An example might be a `REPEAT` ...
使用宏定义的本地标签有时不够强大: 有时您希望能够在多个宏调用之间共享标签。一个例子可能是 REPEAT..。
`UNTIL` loop, in which the expansion of the `REPEAT` macro would need to be able to refer to a label which the `UNTIL` macro had defined. However, for such a macro you would also want to be able to nest these loops.
`UNTIL` 循环，其中 `REPEAT` 宏的扩展需要能够引用 `UNTIL` 宏定义的标签。然而，对于这样一个宏，你也希望能够嵌套这些循环。

NASM provides this level of power by means of a *context stack*. The preprocessor maintains a stack of *contexts*, each of which is characterized by a name. You add a new context to the stack using the `%push`
NASM 通过上下文堆栈提供这种级别的权力。预处理器维护一个上下文堆栈，每个上下文堆栈都拥有属性名称。你可以使用% push 向堆栈中添加一个新的上下文

directive, and remove one using `%pop`. You can define labels that are local to a particular context on the stack.
指令，并使用% pop 删除一个标签。您可以定义堆栈上特定上下文的本地标签。

4.7.1 `%push` and `%pop`: Creating and Removing Contexts
4.7.1% push 和% pop: 创建和删除上下文

The `%push` directive is used to create a new context and place it on the top of the context stack. `%push` takes an optional argument, which is the name of the context. For example:
使用% push 指令创建新的上下文并将其放置在上下文堆栈的顶部。% push 接受一个可选参数，即上下文的名称。例如:

```
%push    foobar
% push foobar 按钮
```

This pushes a new context called `foobar` on the stack. You can have several contexts on the stack with the same name: they can still be distinguished. If no name is given, the context is unnamed (this is normally used when both the `%push` and the `%pop` are inside a single macro definition.)
这会将一个名为 foobar 的新上下文推送到堆栈上。你可以在堆栈上有几个名字相同的上下文: 它们仍然可以被区分。如果没有提供名称，则上下文是未命名的(通常在% push 和% pop 都位于单个宏定义中时使用)

The directive `%pop`, taking one optional argument, removes the top context from the context stack and destroys it, along with any labels associated with it. If an argument is given, it must match the name of the current context, otherwise it will issue an error.
指令% pop 带有一个可选参数，从上下文堆栈中删除顶部上下文并销毁它，以及与之相关的任何标签。如果给定了一个参数，它必须匹配当前上下文的名称，否则它会发出一个错误。

4.7.2 Context−Local Labels
4.7.2 上下文-本地标签

Just as the usage `%%foo` defines a label which is local to the particular macro call in which it is used, the usage `%$foo` is used to define a label which is local to the context on the top of the context stack. So the `REPEAT` and `UNTIL` example given above could be implemented by means of:
正如用法%% foo 定义一个标签，该标签对于使用它的特定宏调用是本地的，用法% $foo 用于定义一个标签，该标签对于上下文堆栈顶部的上下文是本地的。所以上面给出的 REPEAT 和 UNTIL 例子可以通过以下方式实现:

```
%macro repeat 0
% 宏重复 0

    %push   repeat
    % push repeat 按重复
    %$begin:
    % $开始:

%endmacro
% endmacro

%macro until 1
% 宏直到 1

        j%-1    %$begin
        1% 开始
    %pop
    百分之一

%endmacro
% endmacro
```

and invoked by means of, for example,

并通过例如,

```
        mov     cx,string
        mov cx，字符串
        repeat
        重复
        add     cx,3
        加上 cx，3
        scasb
        结痂
        until   e
        直到 e
```

which would scan every fourth byte of a string in search of the byte in `AL`.
它将扫描字符串的每四个字节以搜索 al 中的字节。

If you need to define, or access, labels local to the context *below* the top one on the stack, you can use `%$$foo`, or `%$$$foo` for the context below that, and so on.
如果需要定义或访问堆栈顶部上下文的本地标签，则可以使用% $$foo 或% $$foo 来定义下面的上下文，依此类推。

4.7.3 Context−Local Single−Line Macros
4.7.3 上下文-本地单行宏

NASM also allows you to define single−line macros which are local to a particular context, in just the same way:
NASM 还允许你以同样的方式定义特定上下文的本地单行宏:

```
%define %$localmac 3
% define% $localmac 3
```

will define the single−line macro `%$localmac` to be local to the top context on the stack. Of course, after a subsequent `%push`, it can then still be accessed by the name `%$$localmac`.
将单行宏% $localmac 定义为堆栈顶部上下文的本地化。当然，在后续的% push 之后，仍然可以通过名称% $$localmac 来访问它。

4.7.4 Context Fall−Through Lookup (deprecated)
4.7.4 Context Fall-Through Lookup (不推荐)

Context fall−through lookup (automatic searching of outer contexts) is a feature that was added in NASM version 0.98.03. Unfortunately, this feature is unintuitive and can result in buggy code that would have otherwise been prevented by NASM's error reporting. As a result, this feature has been *deprecated*. NASM version 2.09 will issue a warning when usage of this *deprecated* feature is detected. Starting with NASM version 2.10, usage of this *deprecated* feature will simply result in an *expression syntax error*.
Context fall-through 查找(自动搜索外部上下文)是 NASM 版本 0.98.03 中添加的一个特性。不幸的是，这个功能是不直观的，可能会导致错误代码，否则会被 NASM 的错误报告阻止。因此，这个功能已经被废弃了。NASM 2.09 版会在检测到这个不再使用的功能时发出警告。从 NASM 2.10 版开始，使用这个废弃的特性只会导致一个表达式语法错误。

An example usage of this *deprecated* feature follows:
下面是一个使用这个过时功能的例子:

```
%macro ctxthru 0
% 宏 ctxthrough 0
%push ctx1
% push ctx1
    %assign %$external 1
    % 分配% 外部 1
        %push ctx2
        % push ctx2
            %assign %$internal 1
            % 分配% $内部 1
            mov eax, %$external
            Mov eax,% $external
            mov eax, %$internal
            电影,% $内部
        %pop
        百分之一
%pop
百分之一
%endmacro
% endmacro
```

As demonstrated, `%$external` is being defined in the `ctx1` context and referenced within the `ctx2` context. With context fall−through lookup, referencing an undefined context−local macro like this implicitly searches through all outer contexts until a match is made or isn't found in any context. As a result, `%$external` referenced within the `ctx2` context would implicitly use `%$external` as defined in `ctx1`. Most people would expect NASM to issue an error in this situation because `%$external` was never defined within `ctx2` and also isn't qualified with the proper context depth, `%$$external`.
正如演示的那样,% $external 在 ctx1 上下文中定义，并在 ctx2 上下文中引用。使用上下文下降-通过查找，引用未定义的上下文-本地宏像这样隐式地搜索所有外部上下文，直到匹配完成或在任何上下文中找不到匹配。因此，在 ctx2 上下文中引用的% $external 将隐式使用在 ctx1 中定义的% $external。大多数人希望 NASM 在这种情况下发出一个错误，因为% $external 从未在 ctx2 中定义，也没有使用适当的上下文深度% $external 进行限定。

Here is a revision of the above example with proper context depth:
下面是上面例子的一个修订版，它包含了适当的上下文深度:

```
%macro ctxthru 0
% 宏 ctxthrough 0
%push ctx1
% push ctx1
    %assign %$external 1
    % 分配% 外部 1
        %push ctx2
        % push ctx2
            %assign %$internal 1
            % 分配% $内部 1
            mov eax, %$$external
            Mov eax,% $$external
            mov eax, %$internal
            电影,% $内部
        %pop
        百分之一
%pop
百分之一
%endmacro
% endmacro
```

As demonstrated, `%$external` is still being defined in the `ctx1` context and referenced within the `ctx2` context. However, the reference to `%$external` within `ctx2` has been fully qualified with the proper context depth, `%$$external`, and thus is no longer ambiguous, unintuitive or erroneous.
正如演示的那样,% $external 仍然在 ctx1 上下文中定义，并在 ctx2 上下文中引用。但是，ctx2 中对% $external 的引用已经完全限定了适当的上下文深度,% $external，因此不再模糊、不直观或错误。

4.7.5 `%repl`: Renaming a Context
4.7.5% repl: 重命名上下文

If you need to change the name of the top context on the stack (in order, for example, to have it respond differently to `%ifctx`), you can execute a `%pop` followed by a `%push`; but this will have the side effect of destroying all context−local labels and macros associated with the context that was just popped.
如果需要更改堆栈顶部上下文的名称(例如，为了让它以不同的方式响应% ifctx)，可以执行一个% pop，然后执行一个% push; 但这样做的副作用是破坏所有上下文——与刚刚弹出的上下文相关联的本地标签和宏。

NASM provides the directive `%repl`, which *replaces* a context with a different name, without touching the associated macros and labels. So you could replace the destructive code
NASM 提供了指令% repl，用不同的名称替换上下文，而不触及相关的宏和标签。所以你可以替换破坏性代码

```
%pop
百分之一
%push   newname
% push newname 推送新名字
```

with the non−destructive version `%repl newname`.
使用非破坏性版本% repl newname。

4.7.6 Example Use of the Context Stack: Block IFs
4.7.6 示例使用 Context Stack: Block if

This example makes use of almost all the context−stack features, including the conditional−assembly construct `%ifctx`, to implement a block IF statement as a set of macros.
此示例利用几乎所有的上下文堆栈特性，包括条件程序集构造% ifctx，将块 IF 语句实现为一组宏。

```
%macro if 1
% 宏如果 1

    %push if
    % push if 如果
    j%-1  %$ifnot
    J%-1% $if not

%endmacro
% endmacro

%macro else 0
% macro else 0

  %ifctx if
  如果
        %repl   else
        % repl else
        jmp     %$ifend
        朋友
        %$ifnot:
        % $如果不是:
  %else
  其他%
        %error  "expected ‘if’ before ‘else’"
        % 错误‘如果’在‘其他’之前
  %endif
  % endif

%endmacro
% endmacro

%macro endif 0
% 宏结尾 0

  %ifctx if
  如果
        %$ifnot:
        % $如果不是:
        %pop
        百分之一
  %elifctx      else
  % elifctx else
        %$ifend:
        % 朋友:
        %pop
        百分之一
  %else
  其他%
        %error  "expected ‘if’ or ‘else’ before ‘endif’"
        % error" expected‘ if’or‘ else’before‘ endif’”
  %endif
  % endif

%endmacro
% endmacro
```

This code is more robust than the REPEAT and UNTIL macros given in section 4.7.2, because it uses conditional assembly to check that the macros are issued in the right order (for example, not calling endif before if) and issues a %error if they're not.
这段代码比第 4.7.2 节给出的 REPEAT 和 UNTIL 宏更健壮，因为它使用条件程序集来检查宏是否按正确的顺序发出(例如，在 if 之前不调用 endif)，如果不是，则发出% 错误。

In addition, the endif macro has to be able to cope with the two distinct cases of either directly following an if, or following an else. It achieves this, again, by using conditional assembly to do different things depending on whether the context on top of the stack is if or else.
此外，endif 宏必须能够处理两种截然不同的情况，一种是直接跟随 if，另一种是跟随 else。同样，它通过使用条件汇编来做不同的事情，这取决于堆栈顶部的上下文是 if 还是 else。

The else macro has to preserve the context on the stack, in order to have the %$ifnot referred to by the if macro be the same as the one defined by the endif macro, but has to change the context's name so that endif will know there was an intervening else. It does this by the use of %repl.
Else 宏必须保留堆栈上的上下文，以使 if 宏引用的% $ifnot 与 endif 宏定义的% $ifnot 相同，但必须更改上下文的名称，以便 endif 知道存在一个介入的 else。它通过使用% repl 来实现这一点。

A sample usage of these macros might look like:
这些宏的一个例子可能看起来像:

```
        cmp     ax,bx
        Cmp ax, bx

        if ae
        如果
```

```
          cmp     bx,cx
          Cmp bx，cx

          if ae
          如果
                  mov     ax,cx
                  Mov ax，cx
          else
          其他
                  mov     ax,bx
                  Mov ax，bx
          endif
          译注:

  else
  其他
          cmp     ax,cx
          Cmp ax，cx

          if ae
          如果
                  mov     ax,cx
                  Mov ax，cx
          endif
          译注:

  endif
  译注:
```

The block-`IF` macros handle nesting quite happily, by means of pushing another context, describing the inner `if`, on top of the one describing the outer `if`; thus `else` and `endif` always refer to the last unmatched `if` or `else`.
块 IF 宏通过推送另一个上下文，在描述外部 IF 的上下文之上描述内部 IF 来很好地处理嵌套; 因此 else 和 end 总是指最后一个不匹配的 IF 或 else。

4.8 Stack Relative Preprocessor Directives
4.8 堆栈相对预处理器指令

The following preprocessor directives provide a way to use labels to refer to local variables allocated on the stack.
下面的预处理器指令提供了一种使用标签来引用分配给堆栈的本地变量的方法。

• `%arg` (see section 4.8.1)
% arg (见第 4.8.1 节)

• `%stacksize` (see section 4.8.2)
堆栈大小% (见第 4.8.2 节)

• `%local` (see section 4.8.3)
本地百分比(见第 4.8.3 节)

4.8.1 `%arg` Directive
4.8.1% arg Directive

The `%arg` directive is used to simplify the handling of parameters passed on the stack. Stack based parameter passing is used by many high level languages, including C, C++ and Pascal.
Arg 指令用于简化对堆栈上传递的参数的处理。基于 Stack 的参数传递被许多高级语言使用，包括 c，c + + 和 Pascal。

While NASM has macros which attempt to duplicate this functionality (see section 8.4.5), the syntax is not particularly convenient to use and is not TASM compatible. Here is an example which shows the use of `%arg` without any external macros:
虽然 NASM 有试图复制此功能的宏(参见第 8.4.5 节)，但语法使用起来并不特别方便，并且与 TASM 不兼容。下面是一个没有任何外部宏的% arg 的例子:

```
some_function:
某些功能:

    %push     mycontext        ; save the current context
    % push mycontext; 保存当前上下文
    %stacksize large           ; tell NASM to use bp
    堆栈大小% ; 告诉 NASM 使用 bp
    %arg      i:word, j_ptr:word
    % arg i: word，j _ ptr: word

        mov     ax,[i]
        Mov ax，[ i ]
        mov     bx,[j_ptr]
        Mov bx，[ j _ ptr ]
        add     ax,[bx]
        增加 ax，[ bx ]
        ret
        后悔

    %pop                       ; restore original context
    还原原始上下文
```

This is similar to the procedure defined in section 8.4.5 and adds the value in i to the value pointed to by j_ptr and returns the sum in the ax register. See section 4.7.1 for an explanation of `push` and `pop` and the use of context stacks.
这类似于第 8.4.5 节中定义的过程，将 i 中的值添加到 j _ ptr 指向的值中，并返回 ax 寄存器中的和。请参阅第 4.7.1 节，了解 push 和 pop 以及上下文堆栈的使用。

64

4.8.2 `%stacksize` Directive
4.8.2% 堆栈大小指令

The `%stacksize` directive is used in conjunction with the `%arg` (see section 4.8.1) and the `%local` (see section 4.8.3) directives. It tells NASM the default size to use for subsequent `%arg` and `%local` directives. The `%stacksize` directive takes one required argument which is one of `flat`, `flat64`, `large` or `small`.
% stacksize 指令与% arg (参见第 4.8.1 节)和% local (参见第 4.8.3 节)指令一起使用。它告诉 NASM 后续的% arg 和% local 指令使用的默认大小。% stacksize 指令需要一个必需的参数，即 flat、 flat64、 large 或 small。

```
%stacksize flat
% stacksize flat
```

This form causes NASM to use stack-based parameter addressing relative to `ebp` and it assumes that a near form of call was used to get to this label (i.e. that `eip` is on the stack).
这种形式导致 NASM 相对于 ebp 使用基于堆栈的参数寻址，并且它假设使用了一种近似形式的调用来访问这个标签(即 eip 在堆栈上)。

```
%stacksize flat64
% stack size flat64 堆栈大小
```

This form causes NASM to use stack-based parameter addressing relative to `rbp` and it assumes that a near form of call was used to get to this label (i.e. that `rip` is on the stack).
这种形式导致 NASM 相对于 rbp 使用基于堆栈的参数寻址，并且它假设使用了一种近似形式的调用来到达这个标签(即堆栈上的 rip)。

```
%stacksize large
堆栈大小%
```

This form uses `bp` to do stack-based parameter addressing and assumes that a far form of call was used to get to this address (i.e. that `ip` and `cs` are on the stack).
该表单使用 bp 来执行基于堆栈的参数寻址，并假设使用了一个远程形式的调用来获取这个地址(即 ip 和 cs 在堆栈上)。

```
%stacksize small
堆栈大小小百分比
```

This form also uses `bp` to address stack parameters, but it is different from `large` because it also assumes that the old value of bp is pushed onto the stack (i.e. it expects an `ENTER` instruction). In other words, it expects that `bp`, `ip` and `cs` are on the top of the stack, underneath any local space which may have been allocated by `ENTER`. This form is probably most useful when used in combination with the `%local` directive (see section 4.8.3).
此表单还使用 bp 来寻址堆栈参数，但它与 large 不同，因为它还假定将旧的 bp 值推送到堆栈上(即它期望使用 ENTER 指令)。换句话说，它期望 bp、 ip 和 cs 位于堆栈的顶部，位于 ENTER 可能分配的任何本地空间之下。当与% local 指令结合使用时，这个表单可能是最有用的(参见第 4.8.3 节)。

4.8.3 `%local` Directive
4.8.3% 地方指令

The `%local` directive is used to simplify the use of local temporary stack variables allocated in a stack frame. Automatic local variables in C are an example of this kind of variable. The `%local` directive is most useful when used with the `%stacksize` (see section 4.8.2 and is also compatible with the `%arg` directive (see section 4.8.1). It allows simplified reference to variables on the stack which have been allocated typically by using the `ENTER` instruction. An example of its use is the following:
本地指令% 用于简化在堆栈框架中分配的本地临时堆栈变量的使用。C 语言中的自动局部变量就是这种变量的一个例子。% local 指令在与% stacksize 一起使用时最有用(参见第 4.8.2 节，也与% arg 指令兼容(参见第 4.8.1 节)。它允许简化对堆栈上通常使用 ENTER 指令分配的变量的引用。一个使用它的例子如下:

```
silly_swap:
傻傻的交换:
```

```
%push mycontext                    ; save the current context
% push mycontext                   保存当前上下文
%stacksize small                   ; tell NASM to use bp
堆栈大小小百分比                     告诉 NASM 使用 bp
%assign %$localsize 0              ; see text for explanation
% 分配% $localsize 0                参见文字解释
%local old_ax:word, old_dx:word
% local old _ ax: word，old _ dx: word
     enter     %$localsize,0       ; see text for explanation
     输入       % $localsize，0      参见文字解释
     mov       [old_ax],ax         ; swap ax & bx
     动起来     斧头，斧头            ; 交换斧 & bx
     mov       [old_dx],dx         ; and swap dx & cx
     动起来     [ old _ dx ] ， dx   ; 交换 dx & cx
     mov       ax,bx
     动起来     Ax， bx
     mov       dx,cx
     动起来     Dx， cx
     mov       bx,[old_ax]
     动起来     Bx，[老 _ ax ]
     mov       cx,[old_dx]
     动起来     Cx， [ old _ dx ]
     leave                         ; restore old bp
     走吧                           恢复旧的血压
     ret
     后悔                           ;
%pop                               ; restore original context
百分之一                             ; 恢复原始上下文
```

The `%$localsize` variable is used internally by the `%local` directive and *must* be defined within the current context before the `%local` directive may be used. Failure to do so will result in one expression
% $localsize 变量由% local 指令在内部使用，必须在当前上下文中定义，然后才能使用% local 指令。如果不这样做，将导致一个表达式

syntax error for each `%local` variable declared. It then may be used in the construction of an appropriately sized ENTER instruction as shown in the example.
声明的每个% 本地变量的语法错误。然后，它可以用于构造一个适当大小的 ENTER 指令，如下例所示。

4.9 Reporting User−Defined Errors: `%error`, `%warning`, `%fatal`

4.9 报告用户定义的错误:% 错误,% 警告,% 致命

The preprocessor directive `%error` will cause NASM to report an error if it occurs in assembled code. So if other users are going to try to assemble your source files, you can ensure that they define the right macros by means of code like this:
预处理器指令% error 将导致 NASM 报告一个错误，如果它发生在汇编代码中。因此，如果其他用户试图汇编你的源文件，你可以通过这样的代码来确保他们定义正确的宏:

```
%ifdef F1
% ifdef F1
    ; do some
setup %elifdef F2
    做一些设置%
elifff2
    ; do some different setup
    做一些不同的设置
%else
其他%
    %error "Neither F1 nor F2 was
defined." %endif
    % 错误‰既没有定义 f1 也没有定义 F2。″
```

Then any user who fails to understand the way your code is supposed to be assembled will be quickly warned of their mistake, rather than having to wait until the program crashes on being run and then not knowing what went wrong.
这样，任何不能理解代码组装方式的用户都会很快得到错误的警告，而不必等到程序在运行时崩溃，然后不知道出了什么问题。

Similarly, `%warning` issues a warning, but allows assembly to continue:
类似地,% warning 发出警告，但允许程序集继续:

```
%ifdef F1
% ifdef F1
    ; do some
setup %elifdef F2
    做一些设置%
elifff2
    ; do some different setup
    做一些不同的设置
%else
其他%
    %warning "Neither F1 nor F2 was defined, assuming
    F1." %define F1
    % 警告‰假设 F1，既没有定义 f1 也没有定义 F2。″% 定义 F1
%endif
% endif
```

`%error` and `%warning` are issued only on the final assembly pass. This makes them safe to use in conjunction with tests that depend on symbol values.
% 错误和% 警告仅在最终装配通道上发出。这使得它们可以安全地与依赖于符号值的测试结合使用。

`%fatal` terminates assembly immediately, regardless of pass. This is useful when there is no point in continuing the assembly further, and doing so is likely just going to cause a spew of confusing error messages.

无论通过与否，都会立即终止程序集。当没有必要进一步继续程序集的时候，这是很有用的，而且这样做可能只会导致一连串令人困惑的错误消息。

It is optional for the message string after `%error`, `%warning` or `%fatal` to be quoted. If it is *not*, then single−line macros are expanded in it, which can be used to display more information to the user. For example:
引用% error、% warning 或% fatal 之后的消息字符串是可选的。如果不是，那么单行宏将在其中展开，这可以用来向用户显示更多信息。例如:

```
%if foo > 64
% if foo > 64
    %assign foo_over foo-64
    % 赋予 foo _ 超过 foo-64
    %error foo is foo_over bytes too
large %endif
    % error foo 是 foo _ over bytes 太大
```

4.10 Other Preprocessor Directives
4.10 其他预处理器指令

4.10.1 `%line` Directive
4.10.1% 线性指令

The `%line` directive is used to notify NASM that the input line corresponds to a specific line number in another file. Typically this other file would be an original source file, with the current NASM input being the output of a pre−processor. The `%line` directive allows NASM to output messages which indicate the line number of the original source file, instead of the file that is being read by NASM.
% line 指令用于通知 NASM 输入行对应于另一个文件中的特定行号。通常这个文件是一个原始的源文件，当前的 NASM 输入是一个预处理器的输出。使用% line 指令，NASM 可以输出指示原始源文件行号的消息，而不是 NASM 正在读取的文件。

This preprocessor directive is not generally used directly by programmers, but may be of interest to preprocessor authors. The usage of the `%line` preprocessor directive is as follows:
这个预处理器指令通常不被程序员直接使用，但是预处理器作者可能会感兴趣。% line 预处理器指令的用法如下:

```
%line nnn[+mmm] [filename]
% line nnn [ + mmm ][ filename ]
```

In this directive, `nnn` identifies the line of the original source file which this line corresponds to. `mmm` is an optional parameter which specifies a line increment value; each line of the input file read in is considered to correspond to `mmm` lines of the original source file. Finally, `filename` is an optional parameter which specifies the file name of the original source file.
在这个指令中，nnn 标识了该行对应的原始源文件的行。Mmm 是一个可选参数，它指定一个行增量值; 读入的输入文件的每一行都被认为对应于原始源文件的 mmm 行。最后，filename 是一个可选参数，它指定了原始源文件的文件名。

After reading a `%line` preprocessor directive, NASM will report all file name and line numbers relative to the values specified therein.
读取% 行预处理器指令后，NASM 将报告所有相对于其中指定值的文件名和行号。

If the command line option `--no-line` is given, all `%line` directives are ignored. This may be useful for debugging preprocessed code. See section 2.1.33.
如果命令行选项-无行，所有% 行指令被忽略。这对于调试预处理代码可能很有用。参见 2.1.33 节。

4.10.2 `%!`variable: Read an Environment Variable.
4.10.2% ! 变量: 读取环境变量。

The `%!`*variable* directive makes it possible to read the value of an environment variable at assembly time. This could, for example, be used to store the contents of an environment variable into a string, which could be used at some other point in your code.
% ! Variable 指令使得在装配时读取环境变量的值成为可能。例如，这可以用于将环境变量的内容存储到字符串中，这可以在代码的其他位置使用。

For example, suppose that you have an environment variable `FOO`, and you want the contents of `FOO` to be embedded in your program as a quoted string. You could do that as follows:
例如，假设您有一个环境变量 FOO，并且希望 FOO 的内容以引号字符串的形式嵌入到程序中。你可以这样做:

```
%defstr FOO          %!FOO
% defstr FOO% ! FOO
```

See section 4.1.8 for notes on the `%defstr` directive.
有关% defstr 指令的说明，请参阅第 4.1.8 节。

If the name of the environment variable contains non−identifier characters, you can use string quotes to surround the name of the variable, for example:
如果环境变量的名称包含非标识符字符，则可以使用字符串引号来包围变量的名称，例如:

```
%defstr C_colon      %!'C:'
% defstr c _ colon%
```

4.11 Standard Macros
4.11 标准宏

NASM defines a set of standard macros, which are already defined when it starts to process any source file. If you really need a program to be assembled with no pre−defined macros, you can use the `%clear` directive to empty the preprocessor of everything but context−local preprocessor variables and single−line macros.
NASM 定义了一组标准宏，在开始处理任何源文件时已经定义了这些宏。如果您确实需要在没有预定义宏的情况下组装程序，您可以使用% clear 指令来清空除上下文-本地预处理器变量和单行宏之外的所有预处理器。

Most user−level assembler directives (see chapter 6) are implemented as macros which invoke primitive directives; these are described in chapter 6. The rest of the standard macro set is described here.
大多数用户级汇编指令(见第 6 章)是作为调用原语指令的宏实现的; 这些在第 6 章中描述。标准宏集合的其余部分在这里描述。

4.11.1 NASM Version Macros
4.11.1 NASM 版本宏

The single-line macros `__NASM_MAJOR__`, `__NASM_MINOR__`, `__NASM_SUBMINOR__` and
单行宏 _ NASM _ major _， _ NASM _ minor _， _ NASM _ subminor _ 和

`___NASM_PATCHLEVEL__` expand to the major, minor, subminor and patch level parts of the version number of NASM being used. So, under NASM 0.98.32p1 for example, `__NASM_MAJOR__` would be defined to be 0, `__NASM_MINOR__` would be defined as 98, `__NASM_SUBMINOR__` would be defined to 32, and `___NASM_PATCHLEVEL__` would be defined as 1.
扩展到正在使用的 NASM 版本号的主要、次要、次要和补丁级部分。因此，在 NASM 0.98.32 p1 下，_ NASM _ major _ 将被定义为 0，_ NASM _ minor _ 将被定义为 98，_ NASM _ subminor _ 将被定义为 32，_ NASM _ patchlevel _ 将被定义为 1。

Additionally, the macro `__NASM_SNAPSHOT__` is defined for automatically generated snapshot releases *only*.
另外，宏 _ NASM _ snapshot _ _ 被定义为仅用于自动生成的快照发布。

4.11.2 `__NASM_VERSION_ID__`: NASM Version ID
4.11.2 _ _ NASM _ Version _ ID _ _: NASM 版本 ID

The single-line macro `__NASM_VERSION_ID__` expands to a dword integer representing the full version number of the version of nasm being used. The value is the equivalent to `__NASM_MAJOR__`, `__NASM_MINOR__`, `__NASM_SUBMINOR__` and `___NASM_PATCHLEVEL__` concatenated to produce a single doubleword. Hence, for 0.98.32p1, the returned number would be equivalent to:
单行宏 _ _ NASM _ version _ id _ _ 扩展为一个 dword 整数，表示所使用的 NASM 版本的完整版本号。值相当于 _ _ NASM _ major _ _， _ _ NASM _ minor _， _ _ NASM _ subminor _ 和 _ _ NASM _ patchlevel _ _ 连接起来生成单个双字。因此，对于 0.98.32 p1，返回的数字相当于:

```
dd      0x00622001
老爸老妈，第 0062001 季
```

or
或

```
        db      1,32,98,0
        分贝 1,32,98,0
```

Note that the above lines are generate exactly the same code, the second line is used just to give an indication of the order that the separate values will be present in memory.
请注意，上面的行生成的代码完全相同，第二行只是用来指示单独的值将出现在内存中的顺序。

4.11.3 `__NASM_VER__`: NASM Version string
4.11.3 _ NASM _ ver _ _: NASM Version string

The single−line macro `__NASM_VER__` expands to a string which defines the version number of nasm being used. So, under NASM 0.98.32 for example,
单行宏 _ _ NASM _ ver _ _ 扩展为定义所使用的 NASM 版本号的字符串。例如，在 NASM 0.98.32 下,

```
        db      __NASM_VER__
        数据库 _ NASM _ ver _ _
```

would expand to
会扩展到

```
        db      "0.98.32"
        数据库"0.98.32”
```

4.11.4 `__FILE__` and `__LINE__`: File Name and Line Number
4.11.4 _ _ FILE _ _ 和 _ _ LINE _ _: 文件名和行号

Like the C preprocessor, NASM allows the user to find out the file name and line number containing the current instruction. The macro `__FILE__` expands to a string constant giving the name of the current input file (which may change through the course of assembly if `%include` directives are used), and `__LINE__` expands to a numeric constant giving the current line number in the input file.
像 c 预处理器一样，NASM 允许用户查找包含当前指令的文件名和行号。宏 _ _ FILE _ _ 扩展为一个字符串常量，给出当前输入文件的名称(如果使用% include 指令，则该名称在程序集过程中可能会发生变化)，_ _ LINE _ 扩展为一个数字常量，给出输入文件中的当前行号。

These macros could be used, for example, to communicate debugging information to a macro, since invoking `__LINE__` inside a macro definition (either single−line or multi−line) will return the line number of the macro *call*, rather than *definition*. So to determine where in a piece of code a crash is occurring, for example, one could write a routine `stillhere`, which is passed a line number in `EAX` and outputs something like ‘line 155: still here’. You could then write a macro
例如，这些宏可以用来将调试信息传递给宏，因为在宏定义中调用 _ _ LINE _ (单行或多行)将返回宏调用的行号，而不是定义。例如，为了确定一段代码中崩溃发生的位置，可以在这里编写一个例程 stillhere，它在 EAX 中传递一个行号，然后输出类似“ line 155: still here”的内容。然后你可以写一个宏

```
%macro  notdeadyet 0
% macro notdeadyet 0

        push    eax
        按 eax
        mov     eax,__LINE__
        前进，前进
        call    stillhere
        给我打电话
        pop     eax
        Pop eax

%endmacro
% endmacro
```

and then pepper your code with calls to `notdeadyet` until you find the crash point.
然后在代码中调用 notdead，直到找到崩溃点。

4.11.5 __BITS__: Current BITS Mode
4.11.5 _ BITS _ _: 当前 BITS 模式

The __BITS__ standard macro is updated every time that the BITS mode is set using the BITS XX or [BITS XX] directive, where XX is a valid mode number of 16, 32 or 64. __BITS__ receives the specified mode number and makes it globally available. This can be very useful for those who utilize mode−dependent macros.
每次使用 BITS XX 或[BITS XX]指令设置 BITS 模式时，都会更新 __ BITS _ _ 标准宏，其中 XX 是 16、32 或 64 的有效模式号。_ _ BITS _ _ 接收指定的模式号并使其全局可用。这对于那些使用依赖模式的宏的人来说是非常有用的。

4.11.6 __OUTPUT_FORMAT__: Current Output Format
4.11.6 _ OUTPUT _ Format _ _: Current OUTPUT Format

The __OUTPUT_FORMAT__ standard macro holds the current output format name, as given by the -f option or NASM's default. Type nasm -hf for a list.
_ _ OUTPUT _ format _ _ 标准宏保存当前输出格式名称，由 -f 选项或 NASM 的默认值给出。输入 nasm-hf 作为列表。

```
%ifidn __OUTPUT_FORMAT__, win32
% ifidn _ _ OUTPUT _ format _ _, win32
 %define NEWLINE 13, 10
 % 定义 NEWLINE 13,10
%elifidn __OUTPUT_FORMAT__, elf32
% elifidn _ _ OUTPUT _ format _ _, elf32
 %define NEWLINE 10
 % 定义 NEWLINE 10
%endif
% endif
```

4.11.7 `__DEBUG_FORMAT__`: Current Debug Format
4.11.7 _ DEBUG _ Format _ _: 当前调试格式

If debugging information generation is enabled, The `__DEBUG_FORMAT__` standard macro holds the current debug format name as specified by the `-F` or `-g` option or the output format default. Type `nasm -f` *output* `y` for a list.
如果启用了调试信息生成，则 _ _ DEBUG _ format _ _ 标准宏保存当前调试格式名称，该名称由 -f 或 -g 选项指定，或输出格式默认值指定。输入 nasm-f 输出 y 作为一个列表。

`__DEBUG_FORMAT__` is not defined if debugging is not enabled, or if the debug format specified is `null`.
如果未启用调试，或者指定的调试格式为 null，则不定义 _ _ DEBUG _ format _ _ _。

4.11.8 Assembly Date and Time Macros
4.11.8 汇编日期和时间宏

NASM provides a variety of macros that represent the timestamp of the assembly session.
NASM 提供了各种表示程序集会话时间戳的宏。

- The `__DATE__` and `__TIME__` macros give the assembly date and time as strings, in ISO 8601 format (`"YYYY-MM-DD"` and `"HH:MM:SS"`, respectively.)
_ DATE _ 和 _ TIME _ 宏以 ISO 8601 格式(分别为“ YYYY-MM-DD”和“ HH: MM: SS”)将组装日期和时间作为字符串给出

- The `__DATE_NUM__` and `__TIME_NUM__` macros give the assembly date and time in numeric form; in the format `YYYYMMDD` and `HHMMSS` respectively.
_ DATE _ num _ 和 _ TIME _ num _ 宏分别以数字形式、 YYYYMMDD 和 HHMMSS 格式给出组装日期和时间。

- The `__UTC_DATE__` and `__UTC_TIME__` macros give the assembly date and time in universal time (UTC) as strings, in ISO 8601 format (`"YYYY-MM-DD"` and `"HH:MM:SS"`, respectively.) If the host platform doesn’t provide UTC time, these macros are undefined.
_ UTC _ date _ 和 _ UTC _ time _ 宏以通用时间(UTC)作为字符串，以 ISO 8601 格式(分别为“ YYYY-MM-DD”和“ HH: MM: SS”)给出组装日期和时间如果主机平台没有提供 UTC 时间，这些宏是未定义的。

- The `__UTC_DATE_NUM__` and `__UTC_TIME_NUM__` macros give the assembly date and time universal time (UTC) in numeric form; in the format `YYYYMMDD` and `HHMMSS` respectively. If the host platform doesn’t provide UTC time, these macros are undefined.
_ UTC _ date _ num _ 和 _ UTC _ time _ num _ 宏分别以数字格式、 YYYYMMDD 格式和 HHMMSS 格式给出汇编日期和时间通用时间(UTC)。如果主机平台没有提供 UTC 时间，这些宏是未定义的。

- The `__POSIX_TIME__` macro is defined as a number containing the number of seconds since the POSIX epoch, 1 January 1970 00:00:00 UTC; excluding any leap seconds. This is computed using UTC time if available on the host platform, otherwise it is computed using the local time as if it was UTC.
_ _ POSIX _ time _ _ 宏定义为一个包含自 POSIX 纪元(1970 年 1 月 1 日 00:00:00 UTC)以来的秒数的数字; 不包括任何闰秒。如果在主机平台上可用，则使用 UTC 时间计算，否则使用本地时间计算，就像它是 UTC 一样。

All instances of time and date macros in the same assembly session produce consistent output. For example, in an assembly session started at 42 seconds after midnight on January 1, 2010 in Moscow (timezone UTC+3) these macros would have the following values, assuming, of course, a properly configured environment with a correct clock:
同一个程序集会话中的所有时间和日期宏实例产生一致的输出。例如，在 2010 年 1 月 1 日莫斯科午夜后 42 秒开始的装配会话(时区 UTC + 3)中，这些宏将具有以下值，当然，前提是具有正确时钟的正确配置环境:

```
__DATE__              "2010-01-01"
日期 2010-01-01
__TIME__              "00:00:42"
时间: 00:00:42
```

```
__DATE_NUM__         20100101
日期: num _ 20100101
__TIME_NUM__         000042
_ TIME _ num _ _ 000042
__UTC_DATE__         "2009-12-31"
协调世界时日期2009-12-31
__UTC_TIME__         "21:00:42"
“21:00:42”
__UTC_DATE_NUM__     20091231
协调世界时日期 num20091231
__UTC_TIME_NUM__     210042
UTC 时间 num _ 210042
__POSIX_TIME__       1262293242
时间 _ _ 1262293242
```

4.11.9 __`USE`_package__: Package Include Test

4.11.9 _ USE _ Package _: Package Include Test

When a standard macro package (see chapter 5) is included with the `%use` directive (see section 4.6.4), a single−line macro of the form `__USE_`*package*`__` is automatically defined. This allows testing if a particular package is invoked or not.
当标准宏包(参见第 5 章)包含在% USE 指令(参见第 4.6.4 节)中时，将自动定义一个 _ _ USE _ package _ _ 形式的单行宏。这允许测试特定的包是否被调用。

For example, if the `altreg` package is included (see section 5.1), then the macro `__USE_ALTREG__` is defined.
例如，如果包含 altreg 包(参见第 5.1 节)，则定义宏 _ _ USE _ altreg _ _。

4.11.10 __`PASS`__: Assembly Pass

4.11.10 _ PASS _ _: Assembly PASS

The macro `__PASS__` is defined to be `1` on preparatory passes, and `2` on the final pass. In preprocess−only mode, it is set to `3`, and when running only to generate dependencies (due to the `-M` or `-MG` option, see section 2.1.4) it is set to `0`.
宏 _ _ PASS _ _ 被定义为预备通道为 1，最后通道为 2。在仅进程预处理模式下，它被设置为 3，并且在仅为生成依赖项而运行时(由于 -m 或 -MG 选项，请参阅第 2.1.4 节)，它被设置为 0。

Avoid using this macro if at all possible. It is tremendously easy to generate very strange errors by misusing it, and the semantics may change in future versions of NASM.
尽可能避免使用此宏。错误地使用它很容易产生非常奇怪的错误，语义可能会在未来的 NASM 版本中发生变化。

4.11.11 `STRUC` and `ENDSTRUC`: Declaring Structure Data Types
4.11.11 STRUC 和 ENDSTRUC: 声明结构数据类型

The core of NASM contains no intrinsic means of defining data structures; instead, the preprocessor is sufficiently powerful that data structures can be implemented as a set of macros. The macros `STRUC` and `ENDSTRUC` are used to define a structure data type.
NASM 的核心不包含定义数据结构的内在手段; 相反，预处理器足够强大，数据结构可以作为一组宏来实现。宏 STRUC 和 ENDSTRUC 用于定义结构数据类型。

`STRUC` takes one or two parameters. The first parameter is the name of the data type. The second, optional parameter is the base offset of the structure. The name of the data type is defined as a symbol with the value of the base offset, and the name of the data type with the suffix `_size` appended to it is defined as an `EQU` giving the size of the structure. Once `STRUC` has been issued, you are defining the structure, and should define fields using the `RESB` family of pseudo−instructions, and then invoke `ENDSTRUC` to finish the definition.
`STRUC` 接受一个或两个参数。第一个参数是数据类型的名称。第二个可选参数是结构的基偏移量。数据类型的名称被定义为带有基偏移量值的符号，而带有后缀 `_size` 的数据类型的名称被定义为给出结构大小的 `EQU`。一旦发布了 `STRUC`，您就定义了结构，应该使用 `RESB` 伪指令家族定义字段，然后调用 `ENDSTRUC` 来完成定义。

For example, to define a structure called `mytype` containing a longword, a word, a byte and a string of bytes, you might code
例如，要定义一个名为 mytype 的结构，其中包含一个长单词、一个单词、一个字节和一串字节，您可以编写代码

```
struc   mytype
Struc mytype

  mt_long:      resd    1
  Mt _ long: resd 1
  mt_word:      resw    1
  单词: resw 1
  mt_byte:      resb    1
  Mt _ byte: resb 1
  mt_str:       resb    32
  Mt _ str: resb 32

endstruc
内部结构
```

The above code defines six symbols: `mt_long` as 0 (the offset from the beginning of a `mytype` structure to the longword field), `mt_word` as 4, `mt_byte` as 6, `mt_str` as 7, `mytype_size` as 39, and `mytype` itself as zero.
上面的代码定义了六个符号: mt _ long as 0(从 mytype 结构开始到 longword 字段的偏移量)，mt _ word 为 4，mt _ byte 为 6，mt _ str 为 7，mytype _ size 为 39，mytype 本身为 0。

The reason why the structure type name is defined at zero by default is a side effect of allowing structures to work with the local label mechanism: if your structure members tend to have the same names in more than one structure, you can define the above structure like this:
结构类型名称在默认情况下定义为零的原因是允许结构使用本地标签机制的副作用: 如果您的结构成员倾向于在多个结构中使用相同的名称，您可以像这样定义上述结构:

```
struc mytype
Struc mytype

  .long:        resd    1
  长: 红色 1
  .word:        resw    1
  单词: resw 1
  .byte:        resb    1
  Byte: resb 1
  .str:         resb    32
  Str: resb 32
```

```
endstruc
内部结构
```

This defines the offsets to the structure fields as `mytype.long`, `mytype.word`, `mytype.byte` and `mytype.str`.
这将结构字段的偏移量定义为 mytype.long，mytype.word，mytype.byte 和 mytype.str。

NASM, since it has no *intrinsic* structure support, does not support any form of period notation to refer to the elements of a structure once you have one (except the above local−label notation), so code such as `mov ax,[mystruc.mt_word]` is not valid. `mt_word` is a constant just like any other constant, so the correct syntax is `mov ax,[mystruc+mt_word]` or `mov ax,[mystruc+mytype.word]`.
由于 NASM 没有内在的结构支持，所以它不支持任何形式的周期表示法来引用结构的元素(除了上面的局部标签表示法)，所以像 mov ax [mystruc.mt _ word]这样的代码是无效的。Mt _ word 和其他常量一样是个常量，所以正确的语法是 mov ax，[mystruc + mt _ word]或 mov ax，[mystruc + mytype.word]。

Sometimes you only have the address of the structure displaced by an offset. For example, consider this standard stack frame setup:
有时候你只有结构的地址被一个偏移量替换。例如，考虑下面这个标准的堆栈帧设置:

```
push ebp
按 ebp
mov ebp, esp
(尤指)
sub esp, 40
潜意识，40
```

In this case, you could access an element by subtracting the offset:
在这种情况下，可以通过减去偏移量来访问元素:

```
mov [ebp - 40 + mytype.word], ax
Mov (ebp-40 + mytype.word)
```

However, if you do not want to repeat this offset, you can use –40 as a base offset:
但是，如果不想重复这个偏移量，可以使用 -40 作为基准偏移量:

```
struc mytype, -40
Struc mytype,-40
```

And access an element this way:
以这种方式访问元素:

```
mov [ebp + mytype.word], ax
Mov [ ebp + mytype.word ]
```

4.11.12 `ISTRUC`, `AT` and `IEND`: Declaring Instances of Structures
4.11.12 ISTRUC，AT 和 IEND: 结构实例的声明

Having defined a structure type, the next thing you typically want to do is to declare instances of that structure in your data segment. NASM provides an easy way to do this in the `ISTRUC` mechanism. To declare a structure of type `mytype` in a program, you code something like this:
在定义了一个结构类型之后，你通常要做的下一件事就是在你的数据段中声明这个结构的实例。NASM 在 ISTRUC 机制中提供了一种简单的方法。要在程序中声明一个 mytype 类型的结构，你可以这样编写代码:

```
mystruc:
神秘的:
    istruc mytype
    Istruc mytype

        at mt_long, dd          123456
        在这个长长的地方        123456
        at mt_word, dw
        在这个词，dw            1024
        at   mt_byte, db        'x'
        在   Mt _ byte，db      " x"
        at   mt_str,  db        'hello, world', 13, 10, 0
        在   数据库，数据库     " hello，world"，13,10,0

    iend
    朋友
```

The function of the `AT` macro is to make use of the `TIMES` prefix to advance the assembly position to the correct point for the specified structure field, and then to declare the specified data. Therefore the structure fields must be declared in the same order as they were specified in the structure definition.
AT 宏的功能是利用 TIMES 前缀将程序集位置提前到指定结构字段的正确位置，然后声明指定的数据。因此，结构字段必须按照结构定义中指定的顺序来声明。

If the data to go in a structure field requires more than one source line to specify, the remaining source lines can easily come after the `AT` line. For example:
如果结构字段中的数据需要指定一个以上的源代码行，那么其余的源代码行可以很容易地位于 AT 行之后。例如:

```
      at mt_str,  db      123,134,145,156,167,178,189
    在 mt _ str，db      123,134,145,156,167,178,189
                  db      190,100,0
              数据库      1901000
```

Depending on personal taste, you can also omit the code part of the AT line completely, and start the structure field on the next line:
根据个人喜好，你也可以完全省略 AT 行的代码部分，并在下一行开始结构字段:

```
at mt_str
在大街上
          db      'hello, world'
          Db‘ hello，world'
          db      13,10,0
          分贝 13,10,0
```

4.11.13 ALIGN and ALIGNB: Data Alignment
4.11.13 ALIGN and ALIGN b: 数据对齐

The ALIGN and ALIGNB macros provides a convenient way to align code or data on a word, longword, paragraph or other boundary. (Some assemblers call this directive EVEN.) The syntax of the ALIGN and ALIGNB macros is
ALIGN 和 ALIGNB 宏提供了一种方便的方法来对齐单词、长词、段落或其他边界上的代码或数据。(一些汇编程序称这个指令为 EVENALIGN 和 ALIGNB 宏的语法是

```
align                          ; align on     4–byte boundary
对齐      4                    对齐           4 字节边界
                                          on
align                          ; align    开  16–byte boundary
对齐      16                   对齐       始  16 字节边界
align     8,db 0               ; pad with     0s rather than NOPs
对齐      8，db 0              用; 垫         0 而不是 NOPs
                                                 in
align     4,resb 1               align    to     进  the BSS
对齐      4，resb 1            ; 对齐     对  4  去  生安全服务
alignb                           equivalent      to  previous line
对齐      4                    ; 等效            对  上一行
```

Both macros require their first argument to be a power of two; they both compute the number of additional bytes required to bring the length of the current section up to a multiple of that power of two, and then apply the TIMES prefix to their second argument to perform the alignment.
这两个宏都要求它们的第一个参数是 2 的幂; 它们都计算使当前部分的长度达到 2 的幂的倍数所需的额外字节数，然后将 TIMES 前缀应用于它们的第二个参数以执行对齐。

71

If the second argument is not specified, the default for ALIGN is NOP, and the default for ALIGNB is RESB 1. So if the second argument is specified, the two macros are equivalent. Normally, you can just use ALIGN in code and data sections and ALIGNB in BSS sections, and never need the second argument except for special purposes.
如果没有指定第二个参数，ALIGN 的默认值是 NOP，ALIGNB 的默认值是 resb1。因此，如果指定了第二个参数，两个宏是等价的。通常，你可以在代码和数据部分使用 ALIGN，在 BSS 部分使用 ALIGNB，除非有特殊用途，否则不需要第二个参数。

ALIGN and ALIGNB, being simple macros, perform no error checking: they cannot warn you if their first argument fails to be a power of two, or if their second argument generates more than one byte of code. In each of these cases they will silently do the wrong thing.
ALIGN 和 ALIGNB 作为简单的宏，不执行错误检查：如果它们的第一个参数不是 2 的幂，或者如果它们的第二个参数生成的代码超过一个字节，它们就不能警告您。在每一种情况下，他们都会默默地做错事。

ALIGNB (or ALIGN with a second argument of RESB 1) can be used within structure definitions:
可以在结构定义中使用 ALIGN (或者用 RESB 1 的第二个参数 ALIGN) :

```
struc mytype2
Struc mytype2

  mt_byte:
  字节:
        resb 1
        Resb 1
        alignb 2
        路线 b 2
  mt_word:
  翻译:
        resw 1
        翻译 1
        alignb 4
        校准 b 4
  mt_long:
  很长:
        resd 1
        答案 1
  mt_str:
  译者注:
        resb 32
        回复 32

endstruc
内部结构
```

This will ensure that the structure members are sensibly aligned relative to the base of the structure.
这将确保结构成员相对于结构的底部合理地对齐。

A final caveat: ALIGN and ALIGNB work relative to the beginning of the *section*, not the beginning of the address space in the final executable. Aligning to a 16−byte boundary when the section you’re in is only guaranteed to be aligned to a 4−byte boundary, for example, is a waste of effort. Again, NASM does not check that the section’s alignment characteristics are sensible for the use of ALIGN or ALIGNB.
最后一个警告: ALIGN 和 ALIGNB 的工作相对于节的开头，而不是最终可执行文件中地址空间的开头。例如，当您所在的部分只能保证对齐到 4 字节边界时，对齐到 16 字节边界是一种浪费精力的行为。同样，NASM 不会检查区域的对齐特征是否适合 ALIGN 或 ALIGN b 的使用。

Both ALIGN and ALIGNB do call SECTALIGN macro implicitly. See section 4.11.14 for details.
ALIGN 和 ALIGNB 都隐式调用 SECTALIGN 宏。详见第 4.11.14 节。

See also the smartalign standard macro package, section 5.2.

另请参阅智能对齐标准宏包，第 5.2 节。

4.11.14 `SECTALIGN`: Section Alignment

4.11.14 SECTALIGN: Section Alignment

The `SECTALIGN` macros provides a way to modify alignment attribute of output file section. Unlike the `align=` attribute (which is allowed at section definition only) the `SECTALIGN` macro may be used at any time.
SECTALIGN 宏提供了一种修改输出文件节对齐属性的方法。与 align = 属性不同(只允许在节定义中使用) SECTALIGN 宏可以在任何时候使用。

For example the directive
例如，指令

```
SECTALIGN 16
第 16 节
```

sets the section alignment requirements to 16 bytes. Once increased it can not be decreased, the magnitude may grow only.
将区段对齐要求设置为 16 字节。一旦增加不能减少，幅度可能只增加。

Note that `ALIGN` (see section 4.11.13) calls the `SECTALIGN` macro implicitly so the active section alignment requirements may be updated. This is by default behaviour, if for some reason you want the `ALIGN` do not call `SECTALIGN` at all use the directive
注意 ALIGN (参见第 4.11.13 节)隐式调用 SECTALIGN 宏，以便可以更新活动节对齐要求。这是默认行为，如果出于某种原因，你想要 ALIGN 根本不调用 SECTALIGN，请使用指令

```
SECTALIGN OFF
SECTALIGN 关闭
```

It is still possible to turn in on again by
到... ... 的时候仍然可以重新打开

```
SECTALIGN ON
```

```
SECTALIGN ON
```

72

Chapter 5: Standard Macro Packages
第五章: 标准宏包

The `%use` directive (see section 4.6.4) includes one of the standard macro packages included with the NASM distribution and compiled into the NASM binary. It operates like the `%include` directive (see section 4.6.1), but the included contents is provided by NASM itself.
% use 指令(见第 4.6.4 节)包括 NASM 发行版中包含的一个标准宏包，并编译成 NASM 二进制文件。它的运作方式类似于% include 指令(见第 4.6.1 节)，但包含的头文件是由 NASM 自己提供的。

The names of standard macro packages are case insensitive, and can be quoted or not.
标准宏包的名称不区分大小写，可以引用也可以不引用。

5.1 `altreg`: Alternate Register Names
5.1 altreg: 交替注册名称

The `altreg` standard macro package provides alternate register names. It provides numeric register names for all registers (not just `R8`–`R15`), the Intel−defined aliases `R8L`–`R15L` for the low bytes of register (as opposed to the NASM/AMD standard names `R8B`–`R15B`), and the names `R0H`–`R3H` (by analogy with `R0L`–`R3L`) for `AH`, `CH`, `DH`, and `BH`.
Altreg 标准宏包提供了替代的寄存器名称。它为所有寄存器(不仅仅是 R8-R15)提供数字寄存器名称，为低字节寄存器提供 Intel 定义的别名 R8L-R15L (与 NASM/AMD 标准名称 R8B-R15B 相反)，为 AH、 CH、 DH 和 BH 提供名称 R0H-R3H (与 R0L-R3L 类比)。

Example use:
例子使用:

```
%use altreg
% 使用 altreg

proc:
程序:
      mov r0l,r3h                   ; mov al,bh
      Mov r0l, r3h; mov al, bh
      ret
      后悔
```

See also section 11.1.
参见第 11.1 节。

5.2 `smartalign`: Smart `ALIGN` Macro
5.2 智能对齐: 智能对齐宏

The `smartalign` standard macro package provides for an `ALIGN` macro which is more powerful than the default (and backwards−compatible) one (see section 4.11.13). When the `smartalign` package is enabled, when `ALIGN` is used without a second argument, NASM will generate a sequence of instructions more efficient than a series of `NOP`. Furthermore, if the padding exceeds a specific threshold, then NASM will generate a jump over the entire padding sequence.
Smartalign 标准宏包提供了一个 ALIGN 宏，它比默认的(向后兼容的)宏更强大(参见第 4.11.13 节)。当启用智能对齐包时，当 ALIGN 在没有第二个参数的情况下使用时，NASM 将生成比一系列 NOP 更有效的指令序列。此外，如果填充超过一个特定的阈值，那么 NASM 会在整个填充序列上生成一个跳转。

The specific instructions generated can be controlled with the new `ALIGNMODE` macro. This macro takes two parameters: one mode, and an optional jump threshold override. If (for any reason) you need to turn off the jump completely just set jump threshold value to –1 (or set it to `nojmp`). The following modes are possible:

可以使用新的 ALIGNMODE 宏来控制生成的特定指令。这个宏有两个参数: 一个模式和一个可选的跳跃阈值覆盖。如果(出于任何原因)你需要完全关闭跳转，只需要将跳转阈值设置为 -1(或者设置为 nojmp)。下面的模式是可行的:

- `generic`: Works on all x86 CPUs and should have reasonable performance. The default jump threshold is 8. This is the default.

通用： 适用于所有 x86 处理器，性能合理。默认的跳转阈值是 8。这是默认值。

- `nop`: Pad out with `NOP` instructions. The only difference compared to the standard `ALIGN` macro is that NASM can still jump over a large padding area. The default jump threshold is 16.

带有 NOP 指令的垫子。与标准 ALIGN 宏唯一不同的是，NASM 仍然可以跳过一个很大的填充区域。默认的跳转阈值是 16。

- `k7`: Optimize for the AMD K7 (Athlon/Althon XP). These instructions should still work on all x86 CPUs. The default jump threshold is 16.

K7: AMD k7 的优化(Athlon/Althon XP)。这些指令仍然可以在所有 x86 cpu 上使用。默认的跳转阈值是 16。

- `k8`: Optimize for the AMD K8 (Opteron/Althon 64). These instructions should still work on all x86 CPUs. The default jump threshold is 16.

K8: AMD k8 的优化(Opteron/Althon 64)。这些指令应该仍然可以在所有 x86 cpu 上工作。默认的跳转阈值是 16。

- `p6`: Optimize for Intel CPUs. This uses the long `NOP` instructions first introduced in Pentium Pro. This is incompatible with all CPUs of family 5 or lower, as well as some VIA CPUs and several virtualization solutions. The default jump threshold is 16.

P6: Intel cpu 优化。这使用了 Pentium Pro 中首次引入的较长的 NOP 指令。它不兼容家族 5 或更低版本的所有 cpu，以及一些 VIA cpu 和一些虚拟化解决方案。默认的跳转阈值是 16。

The macro `__ALIGNMODE__` is defined to contain the current alignment mode. A number of other macros beginning with `__ALIGN_` are used internally by this macro package.
宏 _ _ ALIGNMODE _ _ 被定义为包含当前对齐模式。以 _ _ ALIGN _ 开头的许多其他宏在这个宏包内部使用。

73

5.3 fp: Floating-point macros
5.3 fp: 浮点宏

This packages contains the following floating-point convenience macros:
这个包包含以下浮点方便宏:

```
%define Inf             __Infinity__
% 定义 Inf _ _ Infinity _ _
%define NaN             __QNaN__
% 定义 NaN _ QNaN _
%define QNaN            __QNaN__
% 定义 QNaN _ QNaN _
%define SNaN            __SNaN__
% 定义 SNaN _ _ SNaN _ _

%define float8(x)       __float8__(x)
% 定义 float8(x) _ float8 _ (x)
%define float16(x)      __float16__(x)
% 定义 float16(x) _ float16 _ _ (x)
%define float32(x)      __float32__(x)
% 定义 float32(x) _ _ float32 _ _ (x)
%define float64(x)      __float64__(x)
% 定义 float64(x) _ float64 _ (x)
%define float80m(x)     __float80m__(x)
% 定义 float80m (x) _ float80m _ (x)
%define float80e(x)     __float80e__(x)
% 定义 float80e (x) _ float80e _ (x)
%define float128l(x)    __float128l__(x)
% 定义 float128l (x) _ float128l _ (x)
%define float128h(x)    __float128h__(x)
% 定义 float128h (x) _ float128h _ (x)
```

5.4 ifunc: Integer functions
5.4 ifunc: Integer 函数

This package contains a set of macros which implement integer functions. These are actually implemented as special operators, but are most conveniently accessed via this macro package.
这个包包含一组实现整数函数的宏。这些实际上是作为特殊操作符实现的，但是通过这个宏包可以最方便地访问它们。

The macros provided are:
提供的宏包括:

5.4.1 Integer logarithms
5.4.1 整数对数

These functions calculate the integer logarithm base 2 of their argument, considered as an unsigned integer. The only differences between the functions is their respective behavior if the argument provided is not a power of two.
这些函数计算其参数的整数对数基 2，视为无符号整数。如果所提供的参数不是 2 的幂，那么这些函数之间的唯一区别就是它们各自的行为。

The function `ilog2e()` (alias `ilog2()`) generates an error if the argument is not a power of two.
如果参数不是 2 的幂，函数 ilog2e ()(别名 ilog2())会生成一个错误。

The function `ilog2f()` rounds the argument down to the nearest power of two; if the argument is zero it returns zero.
函数 ilog2f ()将参数舍入到 2 的最近幂; 如果参数为零，则返回零。

The function `ilog2c()` rounds the argument up to the nearest power of two.
函数 ilog2c ()将参数四舍五入到 2 的最近幂。

The functions `ilog2fw()` (alias `ilog2w()`) and `ilog2cw()` generate a warning if the argument is not a power of two, but otherwise behaves like `ilog2f()` and `ilog2c()`, respectively.
函数 ilog2fw ()(别名 ilog2w ())和 ilog2cw ()如果参数不是 2 的幂，则会生成一个警告，但在其他情况下，它们的行为分别类似于 ilog2f ()和 ilog2c ()。

Chapter 6: Assembler Directives
第六章: 汇编指令

NASM, though it attempts to avoid the bureaucracy of assemblers like MASM and TASM, is nevertheless forced to support a *few* directives. These are described in this chapter.
尽管 NASM 试图避免像 MASM 和 TASM 这样的装配工的官僚作风，但它仍然被迫支持一些指令。这些将在本章中描述。

NASM's directives come in two types: *user−level* directives and *primitive* directives. Typically, each directive has a user−level form and a primitive form. In almost all cases, we recommend that users use the user−level forms of the directives, which are implemented as macros which call the primitive forms.
NASM 的指令有两种类型: 用户级指令和基本指令。通常，每个指令都有一个用户级的表单和一个基本的表单。在几乎所有的情况下，我们都建议用户使用指令的用户级表单，这些表单被实现为调用原始表单的宏。

Primitive directives are enclosed in square brackets; user−level directives are not.
原语指令用方括号括起来，而用户级指令则不用。

In addition to the universal directives described in this chapter, each object file format can optionally supply extra directives in order to control particular features of that file format. These *format−specific* directives are documented along with the formats that implement them, in chapter 7.
除了本章中描述的通用指令之外，每个对象文件格式还可以选择提供额外的指令，以便控制该文件格式的特定特性。这些特定于格式的指令和实现它们的格式一起在第 7 章中被记录下来。

6.1 `BITS`: Specifying Target Processor Mode
6.1 BITS: 指定目标处理器模式

The `BITS` directive specifies whether NASM should generate code designed to run on a processor operating in 16−bit mode, 32−bit mode or 64−bit mode. The syntax is `BITS XX`, where XX is 16, 32 or 64.
BITS 指令指定 NASM 是否应该生成设计用于在 16 位模式、32 位模式或 64 位模式下运行的处理器上运行的代码。语法是 BITS XX，其中 XX 是 16,32 或 64。

In most cases, you should not need to use `BITS` explicitly. The `aout`, `coff`, `elf`, `macho`, `win32` and `win64` object formats, which are designed for use in 32−bit or 64−bit operating systems, all cause NASM to select 32−bit or 64−bit mode, respectively, by default. The `obj` object format allows you to specify each segment you define as either `USE16` or `USE32`, and NASM will set its operating mode accordingly, so the use of the `BITS` directive is once again unnecessary.
在大多数情况下，你不需要明确地使用 BITS。设计用于 32 位或 64 位操作系统的 about、 coff、 elf、 macho、 win32 和 win64 对象格式，默认情况下都会导致 NASM 分别选择 32 位或 64 位模式。Obj 对象格式允许您指定定义为 use16 或 use32 的每个段，NASM 将相应地设置其操作模式，因此再次不必使用 BITS 指令。

The most likely reason for using the `BITS` directive is to write 32−bit or 64−bit code in a flat binary file; this is because the `bin` output format defaults to 16−bit mode in anticipation of it being used most frequently to write DOS `.COM` programs, DOS `.SYS` device drivers and boot loader software.
使用 BITS 指令最可能的原因是在一个平面二进制文件中编写 32 位或 64 位代码; 这是因为 bin 输出格式默认为 16 位模式，因为预计它将被最频繁地用于编写 DOS。COM 程序，DOS。SYS 设备驱动程序和引导加载程序软件。

The `BITS` directive can also be used to generate code for a different mode than the standard one for the output format.
BITS 指令也可以用来生成不同于输出格式的标准模式的代码。

You do *not* need to specify `BITS 32` merely in order to use 32−bit instructions in a 16−bit DOS program; if you do, the assembler will generate incorrect code because it will be writing code targeted at a 32−bit platform, to be run on a 16−bit one.
您不必仅仅为了在 16 位 DOS 程序中使用 32 位指令而指定 BITS 32; 如果您这样做，汇编程序将生成不正确的代码，因为它将编写针对 32 位平台的代码，以便在 16 位平台上运行。

When NASM is in `BITS 16` mode, instructions which use 32−bit data are prefixed with an 0x66 byte, and those referring to 32−bit addresses have an 0x67 prefix. In `BITS 32` mode, the reverse is true: 32−bit instructions require no prefixes, whereas instructions using 16−bit data need an 0x66 and those working on 16−bit addresses need an 0x67.
当 NASM 处于 BITS 16 模式时，使用 32 位数据的指令前缀为 0x66 字节，而引用 32 位地址的指令前缀为 0x67。在 BITS 32 模式下，情况正好相反: 32 位指令不需要前缀，而使用 16 位数据的指令需要 0x66，而使用 16 位地址的指令需要 0x67。

When NASM is in `BITS 64` mode, most instructions operate the same as they do for `BITS 32` mode.
当 NASM 处于 bits64 模式时，大多数指令的操作方式与 bits32 模式相同。
However, there are 8 more general and SSE registers, and 16−bit addressing is no longer supported.
然而，还有 8 个通用和 SSE 寄存器，16 位寻址不再被支持。

The default address size is 64 bits; 32−bit addressing can be selected with the 0x67 prefix. The default operand size is still 32 bits, however, and the 0x66 prefix selects 16−bit operand size. The `REX` prefix is used both to select 64−bit operand size, and to access the new registers. NASM automatically inserts REX prefixes when necessary.
默认地址大小为 64 位; 可以使用 0x67 前缀选择 32 位地址。然而，默认的操作数大小仍然是 32 位，0x66 前缀选择 16 位操作数大小。REX 前缀既用于选择 64 位操作数大小，也用于访问新的寄存器。必要时，NASM 会自动插入 REX 前缀。

When the `REX` prefix is used, the processor does not know how to address the AH, BH, CH or DH (high 8−bit legacy) registers. Instead, it is possible to access the the low 8−bits of the SP, BP SI and DI registers as SPL, BPL, SIL and DIL, respectively; but only when the REX prefix is used.
当使用 REX 前缀时，处理器不知道如何寻址 AH、 BH、 CH 或 DH (高 8 位遗留)寄存器。相反，可以分别以 SPL、 BPL、 SIL 和 DIL 的形式访问 SP、 BP SI 和 DI 寄存器的低 8 位; 但是只有在使用 REX 前缀时才能访问。

The `BITS` directive has an exactly equivalent primitive form, `[BITS 16]`, `[BITS 32]` and `[BITS 64]`. The user−level form is a macro which has no function other than to call the primitive form.
BITS 指令具有完全等价的原语形式[BITS 16]、[BITS 32]和[BITS 64]。用户级表单是一个宏，除了调用原语表单外没有其他功能。

Note that the space is neccessary, e.g. `BITS32` will *not* work!
注意空间是必需的，比如 bits32 不能工作！

6.1.1 `USE16` & `USE32`: Aliases for BITS
6.1.1 使用 16 & 32: BITS 的别名

The ‘`USE16`’ and ‘`USE32`’ directives can be used in place of ‘`BITS 16`’ and ‘`BITS 32`’, for compatibility with other assemblers.
为了与其他汇编程序兼容，可以使用‘ USE16’和‘ USE32’指令来代替‘ BITS 16’和‘ BITS 32’。

6.2 `DEFAULT`: Change the assembler defaults
6.2 默认值: 更改汇编程序的默认值

The `DEFAULT` directive changes the assembler defaults. Normally, NASM defaults to a mode where the programmer is expected to explicitly specify most features directly. However, this is occasionally obnoxious, as the explicit form is pretty much the only one one wishes to use.
DEFAULT 指令更改汇编程序的默认值。通常情况下，NASM 默认为程序员需要直接明确指定大多数特性的模式。然而，这有时候是令人讨厌的，因为显式的形式几乎是唯一一个希望使用的。

Currently, `DEFAULT` can set `REL` & `ABS` and `BND` & `NOBND`.
目前，DEFAULT 可以设置 REL & ABS 和 BND & NOBND。

6.2.1 `REL` & `ABS`: RIP−relative addressing
6.2.1 REL & ABS: RIP-相对寻址

This sets whether registerless instructions in 64−bit mode are `RIP`–relative or not. By default, they are absolute unless overridden with the `REL` specifier (see section 3.3). However, if `DEFAULT REL` is specified, `REL` is default, unless overridden with the `ABS` specifier, *except when used with an FS or GS segment override*.
这设置 64 位模式下的无寄存器指令是否与 RIP 相关。默认情况下，它们是绝对的，除非被 REL 规范覆盖(参见第 3.3 节)。然而，如果指定了 DEFAULT REL，REL 就是默认的，除非被 ABS 说明重写，除非被 FS 或 GS 段重写。

The special handling of `FS` and `GS` overrides are due to the fact that these registers are generally used as thread pointers or other special functions in 64−bit mode, and generating `RIP`–relative addresses would be extremely confusing.
FS 和 GS 重写的特殊处理是由于这些寄存器通常用作 64 位模式下的线程指针或其他特殊函数，生成 RIP 相对地址将极其混乱。

`DEFAULT REL` is disabled with `DEFAULT ABS`.
`DEFAULT REL 被 DEFAULT ABS 禁用。`

6.2.2 `BND` & `NOBND`: `BND` prefix
6.2.2 BND & NOBND: BND 前缀

If `DEFAULT BND` is set, all bnd−prefix available instructions following this directive are prefixed with bnd. To override it, `NOBND` prefix can be used.
如果默认的 BND 被设置，所有的 BND 前缀可用的指令跟随这个指令是 BND 前缀。要覆盖它，可以使用 NOBND 前缀。

```
DEFAULT BND
违约债券
    call foo            ; BND will be prefixed
    Call foo; BND 将以
    nobnd call foo      ; BND will NOT be prefixed
    Nnd call foo; BND 将不加前缀
```

`DEFAULT NOBND` can disable `DEFAULT BND` and then `BND` prefix will be added only when explicitly specified in code.
`DEFAULT NOBND` 可以禁用 `DEFAULT BND`，然后 `BND` 前缀只有在代码中明确指定时才会被添加。

`DEFAULT BND` is expected to be the normal configuration for writing MPX−enabled code.
`DEFAULT BND` 应该是编写启用 `MPX` 的代码的正常配置。

6.3 SECTION or SEGMENT: Changing and Defining Sections

6.3 SECTION or SEGMENT: Changing and Defining Sections 6.3 区段或段: 更改和定义区段

The `SECTION` directive (`SEGMENT` is an exactly equivalent synonym) changes which section of the output file the code you write will be assembled into. In some object file formats, the number and names of sections are fixed; in others, the user may make up as many as they wish. Hence `SECTION` may sometimes give an error message, or may define a new section, if you try to switch to a section that does not (yet) exist.
SECTION 指令(SEGMENT 是一个完全等效的同义词)更改您编写的代码将汇编到输出文件的哪个部分。在一些目标文件格式中，段的数量和名称是固定的; 在另一些目标文件格式中，用户可以按照自己的意愿组成很多段。因此，如果您尝试切换到一个(尚)不存在的部分，SECTION 有时可能会给出一个错误消息，或者可能会定义一个新的部分。

The Unix object formats, and the `bin` object format (but see section 7.1.3), all support the standardized section names `.text`, `.data` and `.bss` for the code, data and uninitialized−data sections. The `obj` format, by contrast, does not recognize these section names as being special, and indeed will strip off the leading period of any section name that has one.
Unix 对象格式和 bin 对象格式(参见第 7.1.3 节)都支持标准化的章节名称。文本。资料及。代码、数据和未初始化数据部分的 bss。相比之下，obj 格式不承认这些节名是特殊的，而且实际上会去掉任何有节名的前导句点。

6.3.1 The __SECT__ Macro

6.3.1 宏

The `SECTION` directive is unusual in that its user−level form functions differently from its primitive form. The primitive form, `[SECTION xyz]`, simply switches the current target section to the one given. The user−level form, `SECTION xyz`, however, first defines the single−line macro `__SECT__` to be the primitive `[SECTION]` directive which it is about to issue, and then issues it. So the user−level directive
SECTION 指令的不同之处在于，它的用户级表单的功能不同于它的原始表单。原始形式 [SECTION xyz]只是简单地将当前的目标部分切换到给定的部分。但是，用户级表单 SECTION xyz 首先将单行宏 _ _ SECT _ _ 定义为它将要发出的原语[SECTION]指令，然后再发出该指令。所以用户级指令

```
        SECTION .text
        SECTION 文本
```

expands to the two lines
扩展到这两条线

```
%define __SECT__         [SECTION .text]
% define _ _ _ SECT _ [ SECTION. text ]
        [SECTION .text]
        [ SECTION. text ]
```

Users may find it useful to make use of this in their own macros. For example, the `writefile` macro defined in section 4.3.3 can be usefully rewritten in the following more sophisticated form:
用户可能会发现在他们自己的宏中使用这些内容是很有用的。例如，在第 4.3.3 节中定义的 writefile 宏可以用以下更复杂的形式重写:

```
%macro  writefile 2+
% 宏写文件 2 +

        [section .data]
        数据

  %%str:        db      %2
  %% str: db% 2
  %%endstr:
  %% endstr:

        __SECT__
        是的，长官

        mov     dx,%%str
        Mov dx,%% str
        mov     cx,%%endstr-%%str
        Mov cx,%% endstr -% str
        mov     bx,%1
        Mov bx,% 1
        mov     ah,0x40
        移动啊，0 x40
        int     0x21
        Int 0x21 整体 0x21

%endmacro
% endmacro
```

This form of the macro, once passed a string to output, first switches temporarily to the data section of the file, using the primitive form of the `SECTION` directive so as not to modify `__SECT__`. It then declares its string in the data section, and then invokes `__SECT__` to switch back to *whichever* section the user was previously working in. It thus avoids the need, in the previous version of the macro, to include a `JMP` instruction to jump over the data, and also does not fail if, in a complicated `OBJ` format module, the user could potentially be assembling the code in any of several separate code sections.
这种形式的宏一旦将字符串传递给输出，首先使用 SECTION 指令的原始形式临时切换到文件的数据部分，以便不修改 _ _ SECT _ _ _。然后它在数据部分声明它的字符串，然后调用 _ _ SECT _ _ 切换回用户以前工作的部分。因此，在以前的宏版本中，它避免了包含跳过数据的 JMP 指令的需要，而且，如果在一个复杂的 OBJ 格式模块中，用户可能正在将代码组装到几个独立的代码段中的任何一个，它也不会失败。

6.4 `ABSOLUTE`: Defining Absolute Labels
6.4 绝对: 定义绝对标签

The ABSOLUTE directive can be thought of as an alternative form of SECTION: it causes the subsequent code to be directed at no physical section, but at the hypothetical section starting at the given absolute address. The only instructions you can use in this mode are the RESB family.
ABSOLUTE 指令可以被认为是 SECTION 的另一种形式: 它导致后续代码不针对任何物理部分，而是针对从给定的绝对地址开始的假设部分。在这种模式下，你唯一可以使用的指令是 RESB 家族。

ABSOLUTE is used as follows:
ABSOLUTE 的用法如下:

```
absolute 0x1A
绝对 0x1a

        kbuf_chr      resw
        Kbuf _ chr    翻译     1
        kbuf_free     resw
        无拘无束      翻译     1
        kbuf          resw
        Kbuf          翻译     16
```

This example describes a section of the PC BIOS data area, at segment address 0x40: the above code defines kbuf_chr to be 0x1A, kbuf_free to be 0x1C, and kbuf to be 0x1E.
这个例子描述了 PC BIOS 数据区的一部分，在段地址 0x40: 上面的代码定义 kbuf _ chr 为 0x1a，kbuf _ free 为 0x1c，kbuf 为 0x1e。

The user−level form of ABSOLUTE, like that of SECTION, redefines the __SECT__ macro when it is invoked.
ABSOLUTE 的用户级形式，就像 SECTION 一样，在调用 __ SECT _ _ 宏时重新定义 _ _ SECT _ _ 宏。

STRUC and ENDSTRUC are defined as macros which use ABSOLUTE (and also __SECT__). ABSOLUTE doesn’t have to take an absolute constant as an argument: it can take an expression
STRUC 和 ENDSTRUC 被定义为使用 ABSOLUTE (以及 _ _ SECT _)的宏。ABSOLUTE 不需要把一个绝对常量作为参数：它可以采用一个表达式
(actually, a critical expression: see section 3.8) and it can be a value in a segment. For example, a TSR can re−use its setup code as run−time BSS like this:
(实际上，是一个关键表达式: 参见 3.8 节)，它可以是一个段中的值。例如，一个 TSR 可以像这样重用它的设置代码作为运行时 BSS:

```
        org     100h               ; it's a .COM program
        Org 100h; 这是一个 COM 程序

        jmp     setup              ; setup code comes last
        Jmp 安装程序; 安装代码最后出现

        ; the resident part of the TSR goes here
        ; TSR 的常驻部分放在这里
setup:
设置:
        ; now write the code that installs the TSR

here absolute setup

        现在编写安装 TSR 的代码绝对设置

runtimevar1     resw    1
Runtimevar1 resw 1
runtimevar2     resd    20
Runtimevar2 resd 20

tsr_end:
结束:
```

This defines some variables 'on top of' the setup code, so that after the setup has finished running, the space it took up can be re−used as data storage for the running TSR. The symbol 'tsr_end' can be used to calculate the total size of the part of the TSR that needs to be made resident.
这在安装代码之上定义了一些变量，以便在安装程序完成运行之后，它占用的空间可以重新用作运行 TSR 的数据存储。符号' TSR _ end'可以用来计算 TSR 需要驻留的部分的总大小。

6.5 EXTERN: Importing Symbols from Other Modules
6.5 EXTERN: 从其他模块导入符号

EXTERN is similar to the MASM directive EXTRN and the C keyword extern: it is used to declare a symbol which is not defined anywhere in the module being assembled, but is assumed to be defined in some other module and needs to be referred to by this one. Not every object−file format can support external variables: the bin format cannot.
EXTERN 类似于 MASM 指令 EXTRN 和 c 关键字 EXTERN: 它用于声明一个符号，该符号没有在正在组装的模块中的任何地方定义，而是假定在其他一些模块中定义，需要在这个模块中引用。不是每个对象文件格式都能支持外部变量: bin 格式不能。

The EXTERN directive takes as many arguments as you like. Each argument is the name of a symbol:
EXTERN 指令可以接受任意多个参数，每个参数都是一个符号的名称:

```
extern  _printf
外部打印
extern  _sscanf,_fscanf
外部的
```

Some object−file formats provide extra features to the EXTERN directive. In all cases, the extra features are used by suffixing a colon to the symbol name followed by object−format specific text. For example, the obj format allows you to declare that the default segment base of an external should be the group dgroup by means of the directive
一些对象文件格式为 EXTERN 指令提供了额外的特性。在所有情况下，额外的特性都是通过在符号名后面加一个冒号，然后是对象格式特定的文本来实现的。例如，obj 格式允许你通过指令声明一个外部的默认段基应该是组 dgroup

```
extern  _variable:wrt dgroup
Extern _ variable: wrt dgroup
```

The primitive form of `EXTERN` differs from the user−level form only in that it can take only one argument at a time: the support for multiple arguments is implemented at the preprocessor level.
EXTERN 的基本形式与用户级形式的区别仅仅在于它一次只能接受一个参数: 对多个参数的支持是在预处理器级别实现的。

You can declare the same variable as `EXTERN` more than once: NASM will quietly ignore the second and later redeclarations.
您可以不止一次地声明与 EXTERN 相同的变量: NASM 将悄悄地忽略第二次和以后的重新声明。

If a variable is declared both `GLOBAL` and `EXTERN`, or if it is declared as `EXTERN` and then defined, it will be treated as `GLOBAL`. If a variable is declared both as `COMMON` and `EXTERN`, it will be treated as
如果一个变量同时被声明为 GLOBAL 和 EXTERN，或者如果它被声明为 EXTERN 然后被定义，那么它将被视为 GLOBAL。如果一个变量同时声明为 COMMON 和 EXTERN，它将被视为
`COMMON`.
常见。

6.6 `GLOBAL`: Exporting Symbols to Other Modules
6.6 GLOBAL: 将符号导出到其他模块

`GLOBAL` is the other end of `EXTERN`: if one module declares a symbol as `EXTERN` and refers to it, then in order to prevent linker errors, some other module must actually *define* the symbol and declare it as `GLOBAL`. Some assemblers use the name `PUBLIC` for this purpose.
GLOBAL 是 EXTERN 的另一端: 如果一个模块将一个符号声明为 EXTERN 并引用它，那么为了防止链接器错误，其他一些模块必须实际定义该符号并将其声明为 GLOBAL。一些汇编程序为此使用 PUBLIC 这个名字。

`GLOBAL` uses the same syntax as `EXTERN`, except that it must refer to symbols which *are* defined in the same module as the `GLOBAL` directive. For example:
GLOBAL 使用与 EXTERN 相同的语法，除了它必须引用与 GLOBAL 指令在同一模块中定义的符号。例如:

```
global _main
全球主网络
_main:
主要内容:
        ; some code
        一些密码
```

`GLOBAL`, like `EXTERN`, allows object formats to define private extensions by means of a colon. The `elf` object format, for example, lets you specify whether global data items are functions or data:
GLOBAL 与 EXTERN 一样，允许对象格式通过冒号定义私有扩展。例如，elf 对象格式允许你指定全局数据项是函数还是数据：

```
global  hashlookup:function, hashtable:data
全局 hashlookup: function，hashtable: data
```

Like `EXTERN`, the primitive form of `GLOBAL` differs from the user−level form only in that it can take only one argument at a time.
与 EXTERN 一样，GLOBAL 的原始形式与用户级形式的区别只在于它一次只能接受一个参数。

6.7 `COMMON`: Defining Common Data Areas

6.7 COMMON: 定义公共数据区域

The `COMMON` directive is used to declare *common variables*. A common variable is much like a global variable declared in the uninitialized data section, so that
COMMON 指令用于声明公共变量。公共变量很像在未初始化的数据部分中声明的全局变量，所以

```
common  intvar  4
公共因变量 4
```

is similar in function to
在功能上类似于

```
global  intvar
全局整数变量
section .bss
部门 bss

intvar  resd    1
Intvar resd 1
```

The difference is that if more than one module defines the same common variable, then at link time those variables will be *merged*, and references to `intvar` in all modules will point at the same piece of memory.
区别在于，如果多个模块定义了相同的公共变量，那么在链接时，这些变量将被合并，所有模块中对 intvar 的引用将指向同一块内存。

Like `GLOBAL` and `EXTERN`, `COMMON` supports object−format specific extensions. For example, the `obj` format allows common variables to be NEAR or FAR, and the `elf` format allows you to specify the alignment requirements of a common variable:
像 GLOBAL 和 EXTERN 一样，COMMON 支持对象格式的特定扩展。例如，obj 格式允许公共变量为 NEAR 或 FAR，elf 格式允许您指定公共变量的对齐要求:

```
common  commvar  4:near  ; works in OBJ
公共通用词 4: near; 在 OBJ 工作
common  intarray 100:4   ; works in ELF: 4 byte aligned
普通嵌入阵列 100:4; 在 ELF 中工作: 4 字节对齐
```

Once again, like `EXTERN` and `GLOBAL`, the primitive form of `COMMON` differs from the user−level form only in that it can take only one argument at a time.
与 EXTERN 和 GLOBAL 一样，COMMON 的原始形式与用户级形式的不同之处在于，它一次只能接受一个参数。

6.8 `STATIC`: Local Symbols within Modules

6.8 STATIC: 模块中的本地符号

Opposite to `EXTERN` and `GLOBAL`, `STATIC` is local symbol, but should be named according to the global mangling rules (named by analogy with the C keyword `static` as applied to functions or global variables).

与 EXTERN 和 GLOBAL 相反，STATIC 是局部符号，但应该根据全局切割规则命名(与应用于函数或全局变量的 c 关键字 STATIC 类比命名)。

```
static foo
静态 foo
foo:
Foo:
          ; codes
          密码
```

Unlike `GLOBAL`, `STATIC` does not allow object formats to accept private extensions mentioned in section 6.6.
与 GLOBAL 不同，STATIC 不允许对象格式接受第 6.6 节中提到的私有扩展。

6.9 `(G|L)PREFIX`, `(G|L)POSTFIX`: Mangling Symbols

6.9(g | l) PREFIX，(g | l) POSTFIX: Mangling Symbols

`PREFIX`, `GPREFIX`, `LPREFIX`, `POSTFIX`, `GPOSTFIX`, and `LPOSTFIX` directives can prepend or append the given argument to a certain type of symbols. The directive should be as a preprocess statement. Each usage is:
PREFIX、 GPREFIX、 LPREFIX、 POSTFIX、 GPOSTFIX 和 LPOSTFIX 指令可以将给定的参数预置或追加到特定类型的符号。这个指令应该是一个预处理语句。每个用法是:

- `PREFIX`|`GPREFIX`: Prepend the argument to all `EXTERN COMMON`, `STATIC`, and `GLOBAL` symbols

PREFIX | GPREFIX: 将参数预置为所有外部通用、静态和全局符号

- `LPREFIX`: Prepend the argument to all other symbols such as Local Labels, and backend defined symbols

LPREFIX: 将参数预置到所有其他符号，如本地标签和后端定义的符号

- `POSTFIX`|`GPOSTFIX`: Append the argument to all `EXTERN COMMON`, `STATIC`, and `GLOBAL` symbols

POSTFIX | GPOSTFIX: 将参数附加到所有外部通用、静态和全局符号

- `LPOSTFIX`: Append the argument to all other symbols such as Local Labels, and backend defined symbols

`LPOSTFIX`：将参数附加到所有其他符号，如本地标签和后端定义的符号

This is a macro implemented as a `%pragma`:
这是一个作为% 语法实现的宏:

```
%pragma macho lprefix L_
% pragma macho lprefix l _
```

Commandline option is also possible. See also section 2.1.28.
命令行选项也是可能的。参见第 2.1.28 节。

Some toolchains is aware of a particular prefix for its own optimization options, such as code elimination. For instance, Mach−O backend has a linker that uses a simplistic naming scheme to chunk up sections into a meta section. When the `subsections_via_symbols` directive (section 7.8.4) is declared, each symbol is the start of a separate block. The meta section is, then, defined to include sections before the one that starts with a ’L’. `LPREFIX` is useful here to mark all local symbols with the ’L’ prefix to be excluded to the meta section. It converts local symbols compatible with the particular toolchain. Note that local symbols declared with `STATIC` (section 6.8) are excluded from the symbol mangling and also not marked as global.
一些工具链知道自己的优化选项的特定前缀，比如代码消除。例如，Mach-o 后端有一个链接器，它使用一个简单的命名方案来将各个部分分割成一个元部分。当 _ via _ symbols 指令(第 7.8.4 节)被声明时，每个符号都是一个独立块的开始。然后，元部分被定义为包含以“ l”开头的部分之前的部分。在这里，LPREFIX 用于标记所有带有“ l”前缀的本地符号，这些符号将被排除在 meta 部分之外。它转换与特定工具链兼容的本地符号。请注意，使用 STATIC (第 6.8 节)声明的本地符号被排除在损坏的符号之外，也没有标记为全局的。

6.10 `OUTPUT`, `DEBUG`: Generic Namespaces
6.10 OUTPUT，DEBUG: Generic Namespaces

`OUTPUT` and `DEBUG` are generic `%pragma` namespaces that are supposed to redirect to the current output and debug formats. For example, when mangling local symbols via the generic namespace:
`OUTPUT` 和 `DEBUG` 是通用的`% pragma` 命名空间，应该重定向到当前的输出和调试格式。例如，当通过泛型命名空间来处理本地符号时：

```
%pragma output gprefix _
杂注输出 gprefix _%
```

This is useful when the directive is needed to be output format agnostic.
当指令需要不依赖于输出格式时，这很有用。

The example is also euquivalent to this, when the output format is `elf`:
当输出格式为 elf 时，这个例子也相当于这样:

```
%pragma elf gprefix _
% pragma elf gprefix _
```

6.11 `CPU`: Defining CPU Dependencies
6.11 CPU: 定义 CPU 依赖关系

The `CPU` directive restricts assembly to those instructions which are available on the specified CPU.
CPU 指令将汇编限制在指定 CPU 上可用的指令上。

Options are:
选择如下:

- `CPU 8086` Assemble only 8086 instruction set

`CPU 8086` 只汇编 `8086` 指令集

- `CPU 186` Assemble instructions up to the 80186 instruction set

`CPU 186` 汇编指令直到 `80186` 指令集

- `CPU 286` Assemble instructions up to the 286 instruction set

CPU 286 将指令集合到 286 指令集

- `CPU 386` Assemble instructions up to the 386 instruction set

CPU 386 汇编指令到 386 指令集

- `CPU 486` 486 instruction set

CPU 486486 指令集

- `CPU 586` Pentium instruction set

CPU 586 Pentium 指令集

- `CPU PENTIUM` Same as 586

CPU 奔腾与 586 相同

- `CPU 686` P6 instruction set

CPU 686p6 指令集

- `CPU PPRO` Same as 686

CPU PPRO 与 686 相同

- `CPU P2` Same as 686

CPU p2 和 686 一样

- `CPU P3` Pentium III (Katmai) instruction sets

CPU P3 Pentium III (Katmai)指令集

- `CPU KATMAI` Same as P3

中央处理器 KATMAI 和 p3 一样

- `CPU P4` Pentium 4 (Willamette) instruction set

CPU P4 Pentium 4(Willamette)指令集

- `CPU WILLAMETTE` Same as P4

中央处理器 WILLAMETTE 和 p4 一样

- `CPU PRESCOTT` Prescott instruction set

普雷斯科特指令集

- `CPU X64` x86−64 (x64/AMD64/Intel 64) instruction set

CPU X64 x86-64(X64/AMD64/Intel 64)指令集

- `CPU IA64` IA64 CPU (in x86 mode) instruction set

CPU ia64ia64cpu (x86 模式)指令集

All options are case insensitive. All instructions will be selected only if they apply to the selected CPU or lower. By default, all instructions are available.
所有选项都不区分大小写。所有指令只有在应用于所选 CPU 或更低的 CPU 时才会被选中。默认情况下，所有指令都是可用的。

6.12 `FLOAT`: Handling of floating−point constants

6.12 FLOAT: 处理浮点常量

By default, floating−point constants are rounded to nearest, and IEEE denormals are supported. The following options can be set to alter this behaviour:
默认情况下，浮点常量四舍五入到最接近的值，并且支持 IEEE 非正规值。可以设置以下选项来改变这种行为:

- `FLOAT DAZ` Flush denormals to zero

将反法线同花齐放到零

- `FLOAT NODAZ` Do not flush denormals to zero (default)

FLOAT NODAZ 不要将反法线刷到零(默认值)

- `FLOAT NEAR` Round to nearest (default)

向最近点浮动(默认值)

- `FLOAT UP` Round up (toward +Infinity)

向上四舍五入(朝向 + Infinity)

- `FLOAT DOWN` Round down (toward –Infinity)

向下四舍五入(朝向无穷大)

- `FLOAT ZERO` Round toward zero

向零漂浮

- `FLOAT DEFAULT` Restore default settings

还原默认设置

The standard macros `__FLOAT_DAZ__`, `__FLOAT_ROUND__`, and `__FLOAT__` contain the current state, as long as the programmer has avoided the use of the brackeded primitive form, (`[FLOAT]`).
标准宏 _ _ FLOAT _ daz _ _、 _ _ FLOAT _ round _ 和 _ _ FLOAT _ 包含当前状态，只要程序员避免使用带括号的原语形式([FLOAT])。

`__FLOAT__` contains the full set of floating−point settings; this value can be saved away and invoked later to restore the setting.
_ _ FLOAT _ _ 包含完整的浮点设置集；此值可以保存并在以后调用以恢复该设置。

6.13 `[WARNING]`: Enable or disable warnings

6.13[警告] : 启用或禁用警告

The `[WARNING]` directive can be used to enable or disable classes of warnings in the same way as the `-w` option, see section 2.1.25 for more details about warning classes.
[WARNING]指令可用于启用或禁用警告类，其方式与 -w 选项相同，有关警告类的详细信息，请参阅第 2.1.25 节。

- `[warning +`*warning−class*`]` enables warnings for *warning−class*.

[warning + warning-class]为 warning-class 启用警告。

- `[warning −`*warning−class*`]` disables warnings for *warning−class*.

[warning-warning-class]禁用 warning-class 的警告。

- `[warning *`*warning−class*`]` restores *warning−class* to the original value, either the default value or as specified on the command line.
`[ warning * warning-class ]`将 `warning-class` 恢复为原始值，即默认值或命令行中指定的值。

The `[WARNING]` directive also accepts the `all`, `error` and `error=`*warning−class* specifiers.
[WARNING]指令还接受 all、 error 和 error = WARNING 类说明符。

No "user form" (without the brackets) currently exists.
目前没有“用户表单”(没有括号)存在。

82

Chapter 7: Output Formats
第七章: 输出格式

NASM is a portable assembler, designed to be able to compile on any ANSI C−supporting platform and produce output to run on a variety of Intel x86 operating systems. For this reason, it has a large number of available output formats, selected using the `−f` option on the NASM command line. Each of these formats, along with its extensions to the base NASM syntax, is detailed in this chapter.
NASM 是一个便携式汇编程序，旨在能够在任何支持 ANSI c 的平台上编译，并产生输出，以运行在各种 Intel x86 操作系统上。出于这个原因，它有大量可用的输出格式，使用 NASM 命令行上的 -f 选项选择。这些格式中的每一种，以及对基本 NASM 语法的扩展，将在本章详细介绍。

As stated in section 2.1.1, NASM chooses a default name for your output file based on the input file name and the chosen output format. This will be generated by removing the extension (`.asm`, `.s`, or whatever you like to use) from the input file name, and substituting an extension defined by the output format. The extensions are given with each format below.
如第 2.1.1 节所述，NASM 根据输入文件名和选择的输出格式为输出文件选择默认名称。这将通过删除扩展名(。.S，或任何你喜欢使用的)，并替换由输出格式定义的扩展名。下面给出了每种格式的扩展名。

7.1 `bin`: Flat−Form Binary Output
7.1 bin: 扁平形式二进制输出

The `bin` format does not produce object files: it generates nothing in the output file except the code you wrote. Such 'pure binary' files are used by MS−DOS: `.COM` executables and `.SYS` device drivers are pure binary files. Pure binary output is also useful for operating system and boot loader development.
Bin 格式不生成目标文件: 它在输出文件中不生成任何东西，除了您编写的代码。这样的"纯二进制"文件被 MS-DOS 使用:。COM 可执行文件和。SYS 设备驱动程序是纯二进制文件。纯二进制输出对于操作系统和引导加载程序的开发也很有用。

The `bin` format supports multiple section names. For details of how NASM handles sections in the `bin` format, see section 7.1.3.
Bin 格式支持多个段名。有关 NASM 如何处理 bin 格式的区段的详细信息，请参阅第 7.1.3 节。

Using the `bin` format puts NASM by default into 16−bit mode (see section 6.1). In order to use `bin` to write 32−bit or 64−bit code, such as an OS kernel, you need to explicitly issue the `BITS 32` or `BITS 64` directive.
使用 bin 格式将 NASM 默认设置为 16 位模式(参见第 6.1 节)。为了使用 bin 来编写 32 位或 64 位代码，例如操作系统内核，您需要显式地发出 BITS 32 或 BITS 64 指令。

`bin` has no default output file name extension: instead, it leaves your file name as it is once the original extension has been removed. Thus, the default is for NASM to assemble `binprog.asm` into a binary file called `binprog`.
Bin 没有默认的输出文件扩展名: 取而代之的是，当原来的扩展名被删除后，它会保留你的文件名。因此，默认情况下 NASM 会将 binprog.asm 组装成一个叫做 binprog 的二进制文件。

7.1.1 ORG: Binary File Program Origin
7.1.1 ORG: 二进制文件程序起源

The `bin` format provides an additional directive to the list given in chapter 6: `ORG`. The function of the `ORG` directive is to specify the origin address which NASM will assume the program begins at when it is loaded into memory.
Bin 格式为第 6 章给出的列表提供了一个额外的指令: ORG。ORG 指令的作用是指定原始地址，NASM 将假定程序在加载到内存时开始。

For example, the following code will generate the longword `0x00000104`:
例如，下面的代码将生成长词 0x000000104:

```
        org     0x100
        Org 0x100
        dd      label
        标签
label:
标签:
```

Unlike the ORG directive provided by MASM−compatible assemblers, which allows you to jump around in the object file and overwrite code you have already generated, NASM’s ORG does exactly what the directive says: *origin*. Its sole function is to specify one offset which is added to all internal address references within the section; it does not permit any of the trickery that MASM’s version does. See section 12.1.3 for further comments.
与与 MASM 兼容的汇编程序提供的 ORG 指令不同，该指令允许您在目标文件中跳转并覆盖已经生成的代码，NASM 的 ORG 完全按照指令所说的做: origin。它的唯一功能是指定一个偏移量，该偏移量被添加到部分内的所有内部地址引用中; 它不允许 MASM 版本所做的任何欺骗。请参阅第 12.1.3 节的进一步评论。

7.1.2 bin Extensions to the SECTION Directive
7.1.2 对 SECTION 指令的扩展

The bin output format extends the SECTION (or SEGMENT) directive to allow you to specify the alignment requirements of segments. This is done by appending the ALIGN qualifier to the end of the section−definition line. For example,
Bin 输出格式扩展了 SECTION (或 SEGMENT)指令，允许您指定段的对齐要求。这是通过在 section-definition 行的末尾追加 ALIGN 限定符来实现的。例如,

```
section .data   align=16
数据对齐 = 16
```

switches to the section .data and also specifies that it must be aligned on a 16−byte boundary.
切换到.data 部分，并且还指定它必须在 16 字节边界上对齐。

The parameter to ALIGN specifies how many low bits of the section start address must be forced to zero. The alignment value given may be any power of two.
ALIGN 参数指定必须强制将节起始地址的低位数设置为零。给定的对齐值可以是 2 的任意幂。

7.1.3 Multisection Support for the `bin` Format
7.1.3 支持垃圾箱格式的多部分

The `bin` format allows the use of multiple sections, of arbitrary names, besides the "known" `.text`, `.data`, and `.bss` names.
Bin 格式允许使用除了“已知”的.text、 .data 和.bss 名称之外的任意名称的多个部分。

- Sections may be designated `progbits` or `nobits`. Default is `progbits` (except `.bss`, which defaults to `nobits`, of course).

Section 可以是指定的 progbits 或 nobits。 Default 是 progbits (除了.bss，当然是 nobits)。

- Sections can be aligned at a specified boundary following the previous section with `align=`, or at an arbitrary byte−granular position with `start=`.

Sections 可以使用 align = 在前一节之后的指定边界处对齐，也可以使用 start = 在任意的字节粒度位置对齐。

- Sections can be given a virtual start address, which will be used for the calculation of all memory references within that section with `vstart=`.

Section 可以被赋予一个虚拟的开始地址，这个虚拟的开始地址将被用来计算该 section 中的所有内存引用。

- Sections can be ordered using `follows=<section>` or `vfollows=<section>` as an alternative to specifying an explicit start address.

可以使用 follows = < section > 或 vfollows = < section > 对 Sections 进行排序，作为指定显式起始地址的替代方法。

- Arguments to `org`, `start`, `vstart`, and `align=` are critical expressions. See section 3.8. E.g.

Org、 start、 vstart 和 align = 的参数是关键表达式。参见第 3.8 节。
`align=(1 << ALIGN_SHIFT) – ALIGN_SHIFT` must be defined before it is used here.
`ALIGN = (1 < < ALIGN _ shift)-ALIGN _ shift` 必须在使用之前定义。

- Any code which comes before an explicit `SECTION` directive is directed by default into the `.text` section.

任何在显式 SECTION 指令之前出现的代码都默认定向到.text 部分。

- If an `ORG` statement is not given, `ORG 0` is used by default.

如果没有给出 ORG 语句，默认情况下使用 org0。

- The `.bss` section will be placed after the last `progbits` section, unless `start=`, `vstart=`, `follows=`, or `vfollows=` has been specified.

Bss 部分将放在最后一个 progbits 部分之后，除非已经指定了 start = ，vstart = ，follows = ，或者 vfollows = 。

- All sections are aligned on dword boundaries, unless a different alignment has been specified.

除非指定了不同的对齐方式，否则所有部分都在 dword 边界上对齐。

- Sections may not overlap.

部分可能不重叠。

- NASM creates the `section.<secname>.start` for each section, which may be used in your code.

NASM 为每个部分创建一个部分. < secname > . start，它可以在代码中使用。

7.1.4 Map Files
7.1.4 Map 文件

Map files can be generated in `-f bin` format by means of the `[map]` option. Map types of `all` (default), `brief`, `sections`, `segments`, or `symbols` may be specified. Output may be directed to `stdout` (default), `stderr`, or a specified file. E.g. `[map symbols myfile.map]`. No "user form" exists, the square brackets must be used.
Map 文件可以通过[Map]选项以 -f bin 格式生成。Map 类型的所有(默认)，简短，部分，片段，或符号可以指定。Output 可以被定向到 stdout (默认)，stderr，或者一个指定的文件。例如[映射符号 myfile.map]。不存在“用户表单”，必须使用方括号。

7.2 `ith`: Intel Hex Output
第 7.2 条: 英特尔十六进制输出

The `ith` file format produces Intel hex−format files. Just as the `bin` format, this is a flat memory image format with no support for relocation or linking. It is usually used with ROM programmers and similar utilities.
第一种文件格式产生英特尔十六进制格式的文件。就像 bin 格式一样，这是一种平面内存映像格式，不支持重定位或链接。它通常用于 ROM 程序员和类似的实用程序。

All extensions supported by the `bin` file format is also supported by the `ith` file format.
所有支持 bin 文件格式的扩展也支持 ith 文件格式。

`ith` provides a default output file−name extension of `.ith`.
`Ith` 提供默认的输出文件扩展名`.ith`。

7.3 `srec`: Motorola S−Records Output
7.3 srec: Motorola s-Records Output

The `srec` file format produces Motorola S−records files. Just as the `bin` format, this is a flat memory image format with no support for relocation or linking. It is usually used with ROM programmers and similar utilities.
Srec 文件格式生成摩托罗拉 s 记录文件。和 bin 格式一样，这是一种平面内存映像格式，不支持重定位或链接。它通常用于 ROM 程序员和类似的实用程序。

All extensions supported by the `bin` file format is also supported by the `srec` file format.
所有支持 bin 文件格式的扩展也支持 srec 文件格式。

`srec` provides a default output file−name extension of `.srec`.
`Srec` 提供默认的输出文件扩展名`.srec`。

7.4 `obj`: Microsoft OMF Object Files
7.4 obj: microsoftomf 对象文件

The `obj` file format (NASM calls it `obj` rather than `omf` for historical reasons) is the one produced by MASM and TASM, which is typically fed to 16−bit DOS linkers to produce `.EXE` files. It is also the format used by OS/2.
Obj 文件格式(由于历史原因，NASM 称其为 obj 而不是 omf)是 MASM 和 TASM 生成的格式，通常提供给 16 位 DOS 链接器生成。EXE 文件。它也是 OS/2 使用的格式。

`obj` provides a default output file−name extension of `.obj`.
`Obj` 提供默认的输出文件扩展名`.obj`。

`obj` is not exclusively a 16−bit format, though: NASM has full support for the 32−bit extensions to the format. In particular, 32−bit `obj` format files are used by Borland's Win32 compilers, instead of using Microsoft's newer `win32` object file format.
`Obj` 不仅仅是一种 `16` 位的格式，尽管：`NASM` 完全支持这种格式的 `32` 位扩展。特别是，`32` 位 `obj` 格式文件被 `Borland` 的 `win32` 编译器使用，而不是使用微软更新的 `win32` 对象文件格式。

The `obj` format does not define any special segment names: you can call your segments anything you like. Typical names for segments in `obj` format files are `CODE`, `DATA` and `BSS`.
Obj 格式没有定义任何特殊的段名称: 您可以随意调用段名称。对象格式文件中段的典型名称是 CODE、 DATA 和 BSS。

If your source file contains code before specifying an explicit `SEGMENT` directive, then NASM will invent its own segment called `__NASMDEFSEG` for you.
如果你的源文件在指定明确的段指令之前包含代码，那么 NASM 将为你发明它自己的段 _ _ NASMDEFSEG。

When you define a segment in an `obj` file, NASM defines the segment name as a symbol as well, so that you can access the segment address of the segment. So, for example:
当您在 obj 文件中定义一个段时，NASM 也将该段名称定义为一个符号，以便您可以访问该段的段地址。例如:

```
segment data
段数据

dvar:     dw
德瓦尔:   Dw        1234
segment   code
片段      代码
function:
功能:
          mov       ax,data            ; get segment address of data
          动起来    Ax，数据           ; 获取数据的段地址
          mov       ds,ax              ; and move it into DS
          动起来    Ds ax              然后把它移到 DS
          inc
          股份有    word [dvar]        ; now this reference will work
          限公司    Word [ dvar ]      现在这个参考可以工作了
          ret
          后悔
```

The `obj` format also enables the use of the `SEG` and `WRT` operators, so that you can write code which does things like
Obj 格式还允许使用 SEG 和 WRT 操作符，这样你就可以编写代码

```
extern  foo
外部 foo
```

```
mov
动起    ax,seg foo              ; get preferred segment of foo
来      Ax，seg foo             得到首选的 foo 片段
mov
动起    ds,ax
来      Ds ax
mov
动起    ax,data                 ; a different segment
来      Ax，数据                不同的部分
mov
动起    es,ax
来      是的，斧头
mov
动起    ax,[ds:foo]             ; this accesses ‘foo’
来      Ax，[ ds: foo ]         ; 这访问“ foo”
mov
动起    [es:foo wrt data],bx    ; so does this
来      [ es: foo wrt data ]， bx    这个也是
```

7.4.1 `obj` Extensions to the `SEGMENT` Directive

7.4.1 obj 对 SEGMENT 指令的扩展

The `obj` output format extends the `SEGMENT` (or `SECTION`) directive to allow you to specify various properties of the segment you are defining. This is done by appending extra qualifiers to the end of the segment−definition line. For example,
Obj 输出格式扩展了 SEGMENT (或 SECTION)指令，允许您指定正在定义的段的各种属性。这是通过在段定义行的末尾附加额外的限定符来实现的。例如,

```
segment code private align=16
段代码 private align = 16
```

defines the segment `code`, but also declares it to be a private segment, and requires that the portion of it described in this code module must be aligned on a 16−byte boundary.
定义段代码，但也声明它是私有段，并要求在此代码模块中描述的部分必须在 16 字节边界上对齐。

The available qualifiers are:
可用的限定词是:

- `PRIVATE`, `PUBLIC`, `COMMON` and `STACK` specify the combination characteristics of the segment. `PRIVATE` segments do not get combined with any others by the linker; `PUBLIC` and `STACK`
`PRIVATE`、 `PUBLIC`、 `COMMON` 和 `STACK` 指定段的组合特征。`PRIVATE` 片段不通过链接器与其他任何片段进行组合; `PUBLIC` 和 `STACK`

segments get concatenated together at link time; and COMMON segments all get overlaid on top of each other rather than stuck end−to−end.
段在链接时被连接在一起; COMMON 段相互叠加而不是端对端的粘连。

- ALIGN is used, as shown above, to specify how many low bits of the segment start address must be forced to zero. The alignment value given may be any power of two from 1 to 4096; in reality, the only values supported are 1, 2, 4, 16, 256 and 4096, so if 8 is specified it will be rounded up to 16, and 32, 64 and 128 will all be rounded up to 256, and so on. Note that alignment to 4096−byte boundaries is a PharLap extension to the format and may not be supported by all linkers.

ALIGN 如上所示，用于指定段起始地址的多少低位必须强制为零。给定的对齐值可以是从 1 到 4096 的任意二次幂；实际上，支持的唯一值是 1、2、4、16、256 和 4096，因此如果指定 8，它将四舍五入到 16,32、64 和 128 将四舍五入到 256，以此类推。请注意，对 4096 字节边界的对齐是该格式的 PharLap 扩展，可能不被所有链接器支持。

- CLASS can be used to specify the segment class; this feature indicates to the linker that segments of the same class should be placed near each other in the output file. The class name can be any word, e.g. CLASS=CODE.

CLASS 可用于指定段类；该特性向链接器指示同一类的段在输出文件中应放置在彼此附近。类名可以是任何单词，例如 CLASS = CODE。

- OVERLAY, like CLASS, is specified with an arbitrary word as an argument, and provides overlay information to an overlay−capable linker.

OVERLAY，就像 CLASS 一样，是用一个任意的单词作为参数来指定的，它为一个具有覆盖能力的链接器提供覆盖信息。

- Segments can be declared as USE16 or USE32, which has the effect of recording the choice in the object file and also ensuring that NASM's default assembly mode when assembling in that segment is 16−bit or 32−bit respectively.

段可以声明为 use16 或 USE32，其作用是在目标文件中记录选择，并确保 NASM 在该段中装配时的默认装配模式分别为 16 位和 32 位。

- When writing OS/2 object files, you should declare 32−bit segments as FLAT, which causes the default segment base for anything in the segment to be the special group FLAT, and also defines the group if it is not already defined.

在编写 OS/2 目标文件时，应该将 32 位段声明为 FLAT，这将导致该段中任何内容的默认段基为特殊组 FLAT，如果尚未定义该组，还应定义该组。

- The obj file format also allows segments to be declared as having a pre−defined absolute segment address, although no linkers are currently known to make sensible use of this feature; nevertheless, NASM allows you to declare a segment such as SEGMENT SCREEN ABSOLUTE=0xB800 if you need to. The ABSOLUTE and ALIGN keywords are mutually exclusive.

Obj 文件格式还允许声明段具有预定义的绝对段地址，尽管目前还没有链接器明智地使用这一特性; 然而，如果需要，NASM 允许您声明段，如 SEGMENT SCREEN ABSOLUTE = 0xB800。ABSOLUTE 和 ALIGN 关键字是相互排斥的。

NASM's default segment attributes are PUBLIC, ALIGN=1, no class, no overlay, and USE16.
NASM 的默认段属性是 PUBLIC，ALIGN = 1，无类，无覆盖，USE16。

7.4.2 GROUP: Defining Groups of Segments
7.4.2 组: 定义分段组

The obj format also allows segments to be grouped, so that a single segment register can be used to refer to all the segments in a group. NASM therefore supplies the GROUP directive, whereby you can code
Obj 格式还允许对片段进行分组，这样一个单独的片段寄存器就可以用来引用组中的所有片段。因此，NASM 提供了 GROUP 指令，通过这个指令你可以编码

```
segment data
段数据

        ; some data
        一些数据
```

```
segment bss
分段 bss

        ; some uninitialized data
        一些未初始化的数据

group dgroup data bss
分组数据
```

which will define a group called `dgroup` to contain the segments `data` and `bss`. Like `SEGMENT`, `GROUP` causes the group name to be defined as a symbol, so that you can refer to a variable `var` in the `data` segment as `var wrt data` or as `var wrt dgroup`, depending on which segment value is currently in your segment register.
它将定义一个名为 dgroup 的组来包含段数据和 bss。与 SEGMENT 类似，GROUP 将组名定义为符号，这样您就可以将数据段中的变量 var 引用为 var wrt 数据或 var wrt dgroup，具体取决于当前段寄存器中的段值。

If you just refer to `var`, however, and `var` is declared in a segment which is part of a group, then NASM will default to giving you the offset of `var` from the beginning of the *group*, not the *segment*. Therefore `SEG var`, also, will return the group base rather than the segment base.
但是，如果你只是引用 var，并且 var 是在一个段中声明的，这个段是一个组的一部分，那么 NASM 将默认给出 var 从组开始的偏移量，而不是段。因此 SEG var 也会返回组基而不是段基。

NASM will allow a segment to be part of more than one group, but will generate a warning if you do this. Variables declared in a segment which is part of more than one group will default to being relative to the first group that was defined to contain the segment.
NASM 允许一个段成为多个组的一部分，但是如果你这样做，它会生成一个警告。在一个段中声明的变量，如果是多个组的一部分，默认是相对于第一个被定义为包含该段的组。

A group does not have to contain any segments; you can still make WRT references to a group which does not contain the variable you are referring to. OS/2, for example, defines the special group FLAT with no segments in it.
一个组不需要包含任何片段; 你仍然可以对一个不包含你所引用的变量的组进行 WRT 引用。例如，OS/2 定义了一个没有段的 FLAT 特殊组。

7.4.3 UPPERCASE: Disabling Case Sensitivity in Output
7.4.3 大写: 在输出中禁用大小写敏感性

Although NASM itself is case sensitive, some OMF linkers are not; therefore it can be useful for NASM to output single−case object files. The UPPERCASE format−specific directive causes all segment, group and symbol names that are written to the object file to be forced to upper case just before being written. Within a source file, NASM is still case−sensitive; but the object file can be written entirely in upper case if desired.
虽然 NASM 本身是区分大小写的，但是一些 OMF 链接器却不是; 因此，对于 NASM 来说，输出单个大小写的对象文件是很有用的。特定于大写格式的指令会导致写入目标文件的所有段、组和符号名称在写入之前都被强制大写。在源文件中，NASM 仍然是区分大小写的; 但是如果需要的话，目标文件可以完全用大写来写。

UPPERCASE is used alone on a line; it requires no parameters.
大写字母在一行中单独使用; 它不需要参数。

7.4.4 IMPORT: Importing DLL Symbols
7.4.4 导入: 导入 DLL 符号

The IMPORT format−specific directive defines a symbol to be imported from a DLL, for use if you are writing a DLL's import library in NASM. You still need to declare the symbol as EXTERN as well as using the IMPORT directive.
IMPORT 格式特定指令定义了从 DLL 导入的符号，以便在用 NASM 编写 DLL 的导入库时使用。你仍然需要将这个符号声明为 EXTERN 以及使用 IMPORT 指令。

The IMPORT directive takes two required parameters, separated by white space, which are (respectively) the name of the symbol you wish to import and the name of the library you wish to import it from. For example:
IMPORT 指令接受两个必需的参数，用空格分隔，它们分别是(分别)要导入的符号的名称和要从其导入的库的名称。例如:

```
import  WSAStartup wsock32.dll
导入 WSAStartup wsock32.dll
```

A third optional parameter gives the name by which the symbol is known in the library you are importing it from, in case this is not the same as the name you wish the symbol to be known by to your code once you have imported it. For example:
第三个可选参数提供了从中导入符号的库中已知符号的名称，以防这个名称与导入后希望代码知道该符号的名称不同。例如:

```
import  asyncsel wsock32.dll WSAAsyncSelect
Import asyncsel wsock32.dll
```

7.4.5 EXPORT: Exporting DLL Symbols
7.4.5 导出: 导出 DLL 符号

The EXPORT format−specific directive defines a global symbol to be exported as a DLL symbol, for use if you are writing a DLL in NASM. You still need to declare the symbol as GLOBAL as well as using the EXPORT directive.
EXPORT 格式特定指令定义一个全局符号作为 DLL 符号导出，以便在用 NASM 编写 DLL 时使用。你仍然需要将这个符号声明为 GLOBAL 并且使用 EXPORT 指令。

EXPORT takes one required parameter, which is the name of the symbol you wish to export, as it was defined in your source file. An optional second parameter (separated by white space from the first) gives the *external* name of the symbol: the name by which you wish the symbol to be

known to programs using the DLL. If this name is the same as the internal name, you may leave the second parameter off.
EXPORT 接受一个必需的参数，这个参数就是你想要导出的符号的名称，正如它在源文件中定义的那样。第二个可选参数(与第一个参数之间用空格分隔)给出了符号的外部名称：您希望使用 DLL 的程序知道该符号的名称。如果这个名字和内部名字相同，你可以关闭第二个参数。

Further parameters can be given to define attributes of the exported symbol. These parameters, like the second, are separated by white space. If further parameters are given, the external name must also be specified, even if it is the same as the internal name. The available attributes are:
可以提供更多的参数来定义导出符号的属性。这些参数，像第二个一样，用空格分隔。如果给出了更多的参数，则必须指定外部名称，即使它与内部名称相同。可用的属性如下:

• `resident` indicates that the exported name is to be kept resident by the system loader. This is an optimisation for frequently used symbols imported by name.
Resident 表示导出的名称由系统加载程序保留为 resident。这是一个经常使用的按名称导入的符号的优化。

• `nodata` indicates that the exported symbol is a function which does not make use of any initialized data.
Nodata 表示导出的符号是一个不使用任何初始化数据的函数。

• `parm=NNN`, where `NNN` is an integer, sets the number of parameter words for the case in which the symbol is a call gate between 32−bit and 16−bit segments.
Parm = NNN，其中 NNN 是一个整数，为符号是 32 位和 16 位段之间的调用门的情况设置参数字的数目。

• An attribute which is just a number indicates that the symbol should be exported with an identifying number (ordinal), and gives the desired number.
只是一个数字的属性表示该符号应该导出一个标识数字(序号)，并给出所需的数字。

For example:
例如:

```
export  myfunc
Export myfunc 出口 myfunc
export  myfunc TheRealMoreFormalLookingFunctionName
导出更正式的函数名称
export  myfunc myfunc 1234  ; export by ordinal
Export myfunc myfunc 1234; export by ordinal
export  myfunc myfunc resident parm=23 nodata
Export myfunc 驻留 parm = 23 nodata
```

7.4.6 `..start`: Defining the Program Entry Point
7.4.6 开始: 定义程序入口点

`OMF` linkers require exactly one of the object files being linked to define the program entry point, where execution will begin when the program is run. If the object file that defines the entry point is assembled using NASM, you specify the entry point by declaring the special symbol `..start` at the point where you wish execution to begin.
`OMF` 链接器正好需要一个被链接的对象文件来定义程序入口点，当程序运行时，将从该入口点开始执行。如果定义入口点的对象文件是用 `NASM` 组装的，你可以通过声明特殊的符号来指定入口点。.从你希望执行的开始点开始。

7.4.7 `obj` Extensions to the `EXTERN` Directive
7.4.7 obj Extensions to the EXTERN Directive 7.4.7 对外部指令的扩展

If you declare an external symbol with the directive
如果使用指令声明外部符号

```
extern  foo
外部 foo
```

then references such as `mov ax,foo` will give you the offset of `foo` from its preferred segment base (as specified in whichever module `foo` is actually defined in). So to access the contents of `foo` you will usually need to do something like
然后引用，如 mov ax，foo 将给出 foo 与其首选段基的偏移量(在实际定义 foo 的模块中指定)。所以要访问 foo 的内容，你通常需要这样做

```
mov      ax,seg foo       ; get   preferred segment base
动起来   Ax，seg foo       得到    优选段基，优选段基
mov      es,ax              move it   into ES
动起来   是的，斧头        ；动起来    变成 ES
                                 use
mov      ax,[es:foo]        and  使    offset ‘foo’ from it
动起来   斧头，斧头        ；及   用    抵消“ foo”
```

This is a little unwieldy, particularly if you know that an external is going to be accessible from a given segment or group, say `dgroup`. So if `DS` already contained `dgroup`, you could simply code
这有点笨拙，特别是如果您知道将从给定的段或组(比如 dgroup)访问外部内容。所以如果 DS 已经包含了 dgroup，你可以简单的编码

```
mov     ax,[foo wrt dgroup]
Mov ax [ foo wrt dgroup ]
```

However, having to type this every time you want to access `foo` can be a pain; so NASM allows you to declare `foo` in the alternative form
但是，每次想要访问 foo 时都必须键入这个代码可能会很麻烦，因此 NASM 允许您以替代形式声明 foo

```
extern  foo:wrt dgroup
Extern foo: wrt dgroup
```

This form causes NASM to pretend that the preferred segment base of `foo` is in fact `dgroup`; so the expression `seg foo` will now return `dgroup`, and the expression `foo` is equivalent to `foo wrt dgroup`.
这种形式导致 NASM 假装 foo 的首选段基实际上是 dgroup; 因此表达式 seg foo 现在将返回 dgroup，而表达式 foo 等价于 foo wrt dgroup。

This default-`WRT` mechanism can be used to make externals appear to be relative to any group or segment in your program. It can also be applied to common variables: see section 7.4.8.
这个默认的 WRT 机制可以用来让外部表示看起来是相对于你程序中的任何组或段的。它也可以应用于常见变量: 参见第 7.4.8 节。

7.4.8 `obj` Extensions to the `COMMON` Directive
7.4.8 obj 扩展到 COMMON 指令

The `obj` format allows common variables to be either near or far; NASM allows you to specify which your variables should be by the use of the syntax
Obj 格式允许公共变量可近可远; NASM 允许你通过使用语法指定你的变量应该是哪个

```
common nearvar 2:near ; 'nearvar' is a near common
common farvar 10:far ; and 'farvar' is far
公共近变量 2: near; ' nearvar'是近公共 farvar 10: far;
' farvar'是 far
```

Far common variables may be greater in size than 64Kb, and so the OMF specification says that they are declared as a number of *elements* of a given size. So a 10-byte far common variable could be declared as ten one-byte elements, five two-byte elements, two five-byte elements or one ten-byte element.
Far 公共变量的大小可能大于 64kb，因此 OMF 规范说，它们被声明为给定大小的若干个元素。所以一个 10 字节的远程公共变量可以声明为 10 个 1 字节元素，5 个 2 字节元素，2 个 5 字节元素或 1 个 10 字节元素。

Some `OMF` linkers require the element size, as well as the variable size, to match when resolving common variables declared in more than one module. Therefore NASM must allow you to specify the element size on your far common variables. This is done by the following syntax:
一些 OMF 链接器在解析多个模块中声明的公共变量时，需要元素大小以及变量大小来匹配。因此，NASM 必须允许你指定元素大小在你的远公共变量。这是通过以下语法实现的:

		10:far				
common	c_5by2	十点钟			two five-byte	elements
普通的	C _ 5by2	方向	5	;	两个 5 字节	元素
		10:far				
common	c_2by5	十点钟			five two-byte	elements
普通的	C _ 2by5	方向	2	;	5 个 2 字节	元素

If no element size is specified, the default is 1. Also, the `FAR` keyword is not required when an element size is specified, since only far commons may have element sizes at all. So the above declarations could equivalently be
如果未指定元素大小，则默认值为 1。另外，在指定元素大小时，不需要使用 FAR 关键字，因为只有 FAR commons 可以有元素大小。所以上面的声明可以等价地为

```
common  c_5by2    10:5      two five-byte    elements
普通的  C _ 5by2  10:5    ; 两个 5 字节     元素
common  c_2by5    10:2      five two-byte    elements
普通的  C _ 2by5  10:2    ; 5 个 2 字节     元素
```

In addition to these extensions, the `COMMON` directive in `obj` also supports default-`WRT` specification like `EXTERN` does (explained in section 7.4.7). So you can also declare things like
除了这些扩展之外，obj 中的 COMMON 指令也像 EXTERN 一样支持默认 WRT 规范(在 7.4.7 节中解释)。所以你也可以声明

```
common  foo     10:wrt dgroup
普通的  译注:   10: wrt dgroup
                16:far
common  bar     16: far    2:wrt data
普通的  酒吧    (远)       2: wrt 数据
                24:wrt
common  baz     24: wrt    data:6
普通的  Baz     24: wrt    资料: 6
```

7.4.9 Embedded File Dependency Information
7.4.9 嵌入式文件依赖信息

Since NASM 2.13.02, `obj` files contain embedded dependency file information. To suppress the generation of dependencies, use
自 nasm2.13.02 以来，obj 文件包含嵌入的依赖项文件信息

```
%pragma obj nodepend
% 杂注 obj nodepend
```

7.5 `win32`: Microsoft Win32 Object Files
7.5 Win32: microsoftwin32 对象文件

The `win32` output format generates Microsoft Win32 object files, suitable for passing to Microsoft linkers such as Visual C++. Note that Borland Win32 compilers do not use this format, but use `obj` instead (see section 7.4).
Win32 输出格式生成 Microsoft win32 目标文件，适合传递给 Microsoft 链接器，如 Visual c + + 。注意 Borland win32 编译器不使用这种格式，而是使用 obj (参见第 7.4 节)。

`win32` provides a default output file-name extension of `.obj`.
`Win32` 提供默认的输出文件扩展名`.obj`。

Note that although Microsoft say that Win32 object files follow the `COFF` (Common Object File Format) standard, the object files produced by Microsoft Win32 compilers are not compatible with COFF linkers such as DJGPP's, and vice versa. This is due to a difference of opinion over the precise semantics of PC-relative relocations. To produce COFF files suitable for DJGPP, use NASM's `coff` output format; conversely, the `coff` format does not produce object files that Win32 linkers can generate correct output from.
注意，尽管 Microsoft 说 win32 对象文件遵循 COFF (通用对象文件格式)标准，但是由 Microsoft win32 编译器生成的对象文件与 COFF 链接器(如 DJGPP)不兼容，反之亦然。这是由于对 PC 相对重定位的精确语义有不同的看法。要生成适合 DJGPP 的 COFF 文件，请使用 NASM 的 COFF 输出格式; 相反，COFF 格式不会生成 win32 链接器可以从中生成正确输出的目标文件。

7.5.1 `win32` Extensions to the `SECTION` Directive

7.5.1 win32 对 SECTION 指令的扩展

Like the `obj` format, `win32` allows you to specify additional information on the `SECTION` directive line, to control the type and properties of sections you declare. Section types and properties are generated automatically by NASM for the standard section names `.text`, `.data` and `.bss`, but may still be overridden by these qualifiers.
像 obj 格式一样，win32 允许你在 SECTION 指令行上指定额外的信息，来控制你声明的节的类型和属性。Section 类型和属性是由 NASM 为标准的 Section 名称自动生成的。文本。资料及。，但仍可能被这些限定符覆盖。

The available qualifiers are:
可用的限定词是:

- `code`, or equivalently `text`, defines the section to be a code section. This marks the section as readable and executable, but not writable, and also indicates to the linker that the type of the section is code.
代码，或者等价的文本，定义了代码段。这标志着这个部分是可读和可执行的，但不是可写的，并且也向链接器表明这个部分的类型是代码。

- `data` and `bss` define the section to be a data section, analogously to `code`. Data sections are marked as readable and writable, but not executable. `data` declares an initialized data section, whereas `bss` declares an uninitialized data section.
`Data` 和 `bss` 将该节定义为一个数据节，类似于代码。数据段被标记为可读可写，但不是可执行的。`Data` 声明一个初始化的数据段，而 `bss` 声明一个未初始化的数据段。

- `rdata` declares an initialized data section that is readable but not writable. Microsoft compilers use this section to place constants in it.
`Rdata` 声明一个可读但不可写的初始化数据段。`Microsoft` 编译器使用这个部分来放置常量。

- `info` defines the section to be an informational section, which is not included in the executable file by the linker, but may (for example) pass information *to* the linker. For example, declaring an `info`–type section called `.drectve` causes the linker to interpret the contents of the section as command−line options.
`Info` 将该节定义为一个信息节，该信息节不包含在链接器的可执行文件中，但可以(例如)将信息传递给链接器。例如，声明一个叫做。`Directve` 使链接器将该节的内容解释为命令行选项。

- `align=`, used with a trailing number as in `obj`, gives the alignment requirements of the section. The maximum you may specify is 64: the Win32 object file format contains no means to request a greater section alignment than this. If alignment is not explicitly specified, the defaults are 16−byte alignment for code sections, 8−byte alignment for rdata sections and 4−byte alignment for data (and BSS) sections. Informational sections get a default alignment of 1 byte (no alignment), though the value does not matter.

Align = 与 obj 中的尾随数字一起使用，给出了该节的对齐要求。你可以指定的最大值是 64：win32 对象文件格式不包含比这个更大的区域对齐请求。如果没有明确指定对方式，默认值是代码段的 16 字节对方式、 rdata 段的 8 字节对方式和数据(和 BSS)段的 4 字节对方式。Informational 部分的默认对齐为 1 字节(无对齐) ，尽管这个值并不重要。

The defaults assumed by NASM if you do not specify the above qualifiers are:
如果你没有指定上面的限定符，NASM 假设的默认值是:

section .text		code	align=16
文本		代码	对齐 = 16
section .data		data	align=4
数据		资料	对齐 = 4
section	.rdata	rdata	align=8
部分	数据	Rdata	Align = 8
section	.bss	bss	align=4
部分	老板	Bss	对齐 = 4

Any other section name is treated by default like `.text`.
默认情况下，任何其他部分的名称都会像.text 一样处理。

7.5.2 `win32`: Safe Structured Exception Handling
7.5.2 win32: 安全结构化异常处理

Among other improvements in Windows XP SP2 and Windows Server 2003 Microsoft has introduced concept of "safe structured exception handling." General idea is to collect handlers' entry points in designated read−only table and have alleged entry point verified against this table prior exception control is passed to the handler. In order for an executable module to be equipped with such "safe exception handler table," all object modules on linker command line has to comply with certain criteria. If one single module among them does not, then the table in question is omitted and above mentioned run−time checks will not be performed for application in question. Table omission is by default silent and therefore can be easily overlooked. One can instruct linker to refuse to produce binary without such table by passing `/safeseh` command line option.
在 windowsxpsp2 和 windowsserver2003 的其他改进中，微软引入了"安全结构化异常处理"的概念一般的想法是在指定的只读表中收集处理程序的入口点，并在异常控制传递给处理程序之前，根据这个表对指定的入口点进行验证。为了使可执行模块配备这样的"安全异常处理程序表"，链接器命令行上的所有对象模块都必须符合某些条件。如果其中一个模块没有，那么就省略了相关的表，上面提到的运行时检查将不会针对相关的应用程序执行。默认情况下，表省略是无声的，因此很容易被忽略。通过传递/safeseh 命令行选项，可以指示链接器拒绝生成没有这种表的二进制文件。

Without regard to this run−time check merits it's natural to expect NASM to be capable of generating modules suitable for `/safeseh` linking. From developer's viewpoint the problem is two−fold:
如果不考虑这种运行时检查的优点，那么很自然地期望 NASM 能够生成适合/safeseh 链接的模块。从开发者的角度来看，问题有两方面:

- how to adapt modules not deploying exception handlers of their own;

如何适应不部署自己的异常处理程序的模块;

- how to adapt/develop modules utilizing custom exception handling;

如何使用自定义异常处理来适应/开发模块;

Former can be easily achieved with any NASM version by adding following line to source code:
Former 可以很容易地用任何 NASM 版本实现，只需在源代码中添加以下行:

```
$@feat.00 equ 1
```

```
$@ feat. 00 equ 1
```

As of version 2.03 NASM adds this absolute symbol automatically. If it’s not already present to be precise. I.e. if for whatever reason developer would choose to assign another value in source file, it would still be perfectly possible.
从 2.03 版本开始，NASM 会自动添加这个绝对符号。如果它还没有准确地显示出来。也就是说，不管出于什么原因，开发人员都会选择在源文件中赋予另一个值，这仍然是完全可能的。

Registering custom exception handler on the other hand requires certain "magic." As of version 2.03 additional directive is implemented, `safeseh`, which instructs the assembler to produce appropriately formatted input data for above mentioned "safe exception handler table." Its typical use would be:
另一方面，注册自定义异常处理程序需要一定的“魔力”在 2.03 版本中，附加的指令 safeseh 被实现，它指示汇编程序为上面提到的“安全异常处理程序表”生成适当格式的输入数据它的典型用法是:

```
section     .text
部分        短信
extern      _MessageBoxA@16
外科医生 邮箱@16
%if         __NASM_VERSION_ID__ >= 0x02030000
如果        = 0x02030000
safeseh     handler              ; register handler as "safe handler"
安全        处理者               ; 将处理程序注册为”安全处理程序”
%endif
% endif
handler:
处理人:
            push     DWORD 1 ; MB_OKCANCEL
            推       DWORD 1 MB _ okcancel
            push     DWORD caption
            推       DWORD 标题
            push     DWORD text
            推       DWORD 文本
            push     DWORD 0
            推       DWORD 0
            call     _MessageBoxA@16
            打电话   邮箱@16
            sub      eax,1      ; incidentally suits as return value
            字幕     Eax，1     顺便说一下，适合作为回报价值
                                ; for exception handler
                                ; 对于异常处理程序
```

```
        ret
        后悔
global  _main
全球    主要
_main:
主要内
容:
                DWOR
                D
        push    DWOR  handler
        推      D     处理者
                DWOR
                D
        push    DWOR  [fs:0]
        推      D     [ fs: 0]
                DWOR
                D
        mov     DWOR  [fs:0],esp ; engage exception handler
        动起来  D     参与异常处理程序
        xor     eax,eax
        Xor     前进，前进
        mov     eax,DWORD[eax]      ; cause exception
        动起来  埃克斯，DWORD       引起例外
                DWOR
                D
        pop     DWOR  [fs:0]        ; disengage exception handler
        啪      D     [ fs: 0]      ; 解除异常处理程序
                esp,4
        add     尤其是
        添加    4
        ret
        后悔
text:   db      'OK to rethrow, CANCEL to generate core dump',0
文字:   数据库  '可以重新抛出，取消生成核心转储'，0
caption:db      'SEGV',0
标题: 数据库    " SEGV"，0
section .drectve info
部分    。方向信息
        db      '/defaultlib:user32.lib /defaultlib:msvcrt.lib '
        数据库  '/defaultlib: user32.lib/defaultlib: msvcrt.lib'
```

As you might imagine, it's perfectly possible to produce .exe binary with "safe exception handler table" and yet engage unregistered exception handler. Indeed, handler is engaged by simply manipulating `[fs:0]` location at run−time, something linker has no power over, run−time that is. It should be explicitly mentioned that such failure to register handler's entry point with `safeseh` directive has undesired side effect at run−time. If exception is raised and unregistered handler is to be executed, the application is abruptly terminated without any notification whatsoever. One can argue that system could at least have logged some kind "non−safe exception handler in x.exe at address n" message in event log, but no, literally no notification is provided and user is left with no clue on what caused application failure.

正如你可能想象的那样，完全有可能生成。使用"安全异常处理程序表"的 exe 二进制文件，同时使用未注册的异常处理程序。事实上，处理程序是通过简单地在运行时操作[fs: 0]位置来实现的，而

链接器对运行时没有任何权限。需要明确指出的是，这种用 safeseh 指令注册处理程序入口点的失败会在运行时产生不希望看到的副作用。如果异常被引发，未注册的处理程序将被执行，应用程序将在没有任何通知的情况下被突然终止。有人可能会说，系统至少可以在事件日志中记录某种“ x.exe at address n”中的“非安全异常处理程序”消息，但是没有，实际上没有提供任何通知，用户也不知道是什么导致了应用程序失败。

Finally, all mentions of linker in this paragraph refer to Microsoft linker version 7.x and later. Presence of `@feat.00` symbol and input data for "safe exception handler table" causes no backward incompatibilities and "safeseh" modules generated by NASM 2.03 and later can still be linked by earlier versions or non−Microsoft linkers.
最后，本段中提到的所有链接器都是指 Microsoft 链接器版本 7.x 及更高版本。存在@feat. 00 符号和”安全异常处理程序表”的输入数据不会导致向后不兼容性，NASM 2.03 及以后生成的“ safeseh” 模块仍然可以通过早期版本或非 Microsoft 链接器链接。

7.5.3 Debugging formats for Windows
7.5.3 Windows 调试格式

The `win32` and `win64` formats support the Microsoft CodeView debugging format. Currently CodeView version 8 format is supported (`cv8`), but newer versions of the CodeView debugger should be able to handle this format as well.
Win32 和 win64 格式支持 microsoftcodeview 调试格式。目前支持 CodeView version 8 格式 (cv8)，但是新版本的 CodeView 调试器也应该能够处理这种格式。

7.6 `win64`: Microsoft Win64 Object Files
7.6 Win64: 微软 win64 对象文件

The `win64` output format generates Microsoft Win64 object files, which is nearly 100% identical to the `win32` object format (section 7.5) with the exception that it is meant to target 64−bit code and the x86−64 platform altogether. This object file is used exactly the same as the `win32` object format (section 7.5), in NASM, with regard to this exception.
Win64 输出格式生成 Microsoft win64 对象文件，它与 win32 对象格式(第 7.5 节)几乎 100% 相同，只是它的目标是 64 位代码和 x86-64 平台。这个目标文件的使用方式和 NASM 中的 win32 目标格式(7.5 节)完全相同。

7.6.1 `win64`: Writing Position−Independent Code
7.6.1 win64: 编写位置无关代码

While `REL` takes good care of RIP−relative addressing, there is one aspect that is easy to overlook for a
尽管 REL 很好地处理了 RIP 相对寻址，但有一个方面很容易被忽视
Win64 programmer: indirect references. Consider a switch dispatch table:
Win64 程序员: 间接引用。考虑一个交换机调度表:

```
        jmp     qword [dsptch+rax*8]
        Qword [ dsptch + rax * 8]
        ...
        ...
dsptch: dq      case0
Dsptch: dq case 0
        dq      case1
        Dq case 1
        ...
        ...
```

Even a novice Win64 assembler programmer will soon realize that the code is not 64−bit savvy. Most notably linker will refuse to link it with
即使是 win64 汇编程序员的新手也会很快意识到代码不是 64 位的。最值得注意的是链接器将拒绝链接它

```
'ADDR32' relocation to '.text' invalid without /LARGEADDRESSAWARE:NO
" ADDR32"重定位到" . text"无效，没有/大型地址软件: 否
```

So [s]he will have to split jmp instruction as following:
因此，他必须将 jmp 指令拆分为以下步骤:

```
        lea     rbx,[rel dsptch]
        [法语]
        jmp     qword [rbx+rax*8]
        [ rbx + rax * 8]
```

What happens behind the scene is that effective address in `lea` is encoded relative to instruction pointer, or in perfectly position−independent manner. But this is only part of the problem! Trouble is that in .dll context `caseN` relocations will make their way to the final module and might have to be adjusted at .dll load time. To be specific when it can't be loaded at preferred address. And when this occurs, pages with such relocations will be rendered private to current process, which kind of undermines the idea of sharing .dll. But no worry, it's trivial to fix:
幕后发生的是 lea 中的有效地址相对于指令指针进行了编码，或者以完全位置无关的方式进行了编码。但这只是问题的一部分！问题在于。在 dll 环境下，卡森重新定位将会进入最终模块，可能需要在。Dll 加载时间。当它不能在首选地址加载时要具体。当这种情况发生时，带有这种重定位的页面将被当前进程私有化，这就有点破坏了共享的想法。Dll.但是不用担心，这个问题很容易解决:

```
        lea     rbx,[rel dsptch]
        [法语]
        add     rbx,[rbx+rax*8]
        加 rbx [ rbx + rax * 8]
        jmp     rbx
        Jmp rbx
        ...
        ...
dsptch: dq      case0-dsptch
Dqcase0-dsptch
        dq      case1-dsptch
        Dq case1-dsptch
        ...
        ...
```

NASM version 2.03 and later provides another alternative, `wrt ..imagebase` operator, which returns offset from base address of the current image, be it .exe or .dll module, therefore the name. For those acquainted with PE−COFF format base address denotes start of `IMAGE_DOS_HEADER` structure. Here is how to implement switch with these image−relative references:
NASM 版本 2.03 及其后的版本提供了另一个替代方案 wrt。.Imagebase 操作符，它从当前图像的基地址返回偏移量。或者。Dll 模块，因此名为。对于那些熟悉 PE-COFF 格式的人来说，基地址表示 IMAGE _ dos _ header 结构的开始。下面是如何使用这些图像相关引用来实现交换:

```
        lea     rbx,[rel dsptch]
        [法语]
        mov     eax,[rbx+rax*4]
        Mov eax, [ rbx + rax * 4]
        sub     rbx,dsptch wrt ..imagebase
        图片库
        add     rbx,rax
        添加 rbx, rax
        jmp     rbx
        Jmp rbx
        ...
        ...
dsptch: dd      case0 wrt ..imagebase
Dsptch: dd case0 wrt. imagebase
        dd      case1 wrt ..imagebase
```

编译: pestwave

One can argue that the operator is redundant. Indeed, snippet before last works just fine with any NASM version and is not even Windows specific... The real reason for implementing `wrt ..imagebase` will become apparent in next paragraph.
人们可以说这个运算符是多余的。事实上，最后一个代码片段在任何 NASM 版本中都能正常工作，甚至不是 Windows 特有的... ... 实现 wrt 的真正原因。.Imagebase 将在下一段显示出来。

It should be noted that `wrt ..imagebase` is defined as 32−bit operand only:
值得注意的是，wrt. imagebase 仅被定义为 32 位操作数:

```
dd
编译:
pestwa    label     wrt ..imagebase                ; ok
ve        标签      图片库                          好的
dq        label     wrt ..imagebase                ; bad
Dq        标签      图片库                          ; 糟糕
          eax,label                                  ok
mov       伊克斯，标  wrt  ..imagebase                好
动起来     签          Wrt  . . 图像库                ;  的
mov       rax,label   wrt  ..imagebase                bad
动起来     Rax 标签    Wrt  . . 图像库                ;  坏
```

7.6.2 `win64`: Structured Exception Handling
7.6.2 win64: 结构化异常处理

Structured exception handing in Win64 is completely different matter from Win32. Upon exception program counter value is noted, and linker−generated table comprising start and end addresses of all the functions [in given executable module] is traversed and compared to the saved program counter. Thus so called `UNWIND_INFO` structure is identified. If it’s not found, then offending subroutine is assumed to be "leaf" and just mentioned lookup procedure is attempted for its caller. In Win64 leaf function is such function that does not call any other function *nor* modifies any Win64 non−volatile registers, including stack pointer. The latter ensures that it’s possible to identify leaf function’s caller by simply pulling the value from the top of the stack.
Win64 中的结构化异常处理与 win32 完全不同。注意到异常程序计数器的值，遍历由所有函数(在给定的可执行模块中)的开始和结束地址组成的链接器生成的表，并与保存的程序计数器进行比较。这样就确定了所谓的 UNWIND _ info 结构。如果没有找到，那么冒犯的子例程被认为是“ leaf”，并且刚才提到的查找过程被尝试用于它的调用者。在 win64 中，leaf 函数不调用任何其他函数，也不修改任何 win64 非易失性寄存器，包括堆栈指针。后者确保通过简单地从堆栈顶部拉取值来识别 leaf 函数的调用者是可能的。

92

While majority of subroutines written in assembler are not calling any other function, requirement for non−volatile registers' immutability leaves developer with not more than 7 registers and no stack frame, which is not necessarily what [s]he counted with. Customarily one would meet the requirement by saving non−volatile registers on stack and restoring them upon return, so what can go wrong? If [and only if] an exception is raised at run−time and no `UNWIND_INFO` structure is associated with such "leaf" function, the stack unwind procedure will expect to find caller's return address on the top of stack immediately followed by its frame. Given that developer pushed caller's non−volatile registers on stack, would the value on top point at some code segment or even addressable space? Well, developer can attempt copying caller's return address to the top of stack and this would actually work in some very specific circumstances. But unless developer can guarantee that these circumstances are always met, it's more appropriate to assume worst case scenario, i.e. stack unwind procedure going berserk. Relevant question is what happens then? Application is abruptly terminated without any notification whatsoever. Just like in Win32 case, one can argue that system could at least have logged "unwind procedure went berserk in x.exe at address n" in event log, but no, no trace of failure is left.
虽然大多数用汇编程序编写的子程序不调用任何其他函数，但对非易失寄存器的不可变性的要求使得开发人员只有不超过 7 个寄存器和没有堆栈帧，这并不一定是他计算的内容。通常人们会通过在堆栈上保存非易失性寄存器并在返回时恢复它们来满足需求，所以会出什么问题呢？如果[且仅当]在运行时引发异常，且没有 UNWIND _ info 结构与这样的" leaf"函数相关联，堆栈展开过程将期望在堆栈顶部找到调用方的返回地址，紧接着是它的帧。假设开发者将调用者的非易失性寄存器放在堆栈上，那么顶部的值会在某个代码段甚至可寻址空间上吗？嗯，开发者可以尝试复制调用者的返回地址到栈的顶部，这实际上会在一些非常特殊的情况下工作。但是，除非开发人员能够保证这些情况总是得到满足，否则更合适的做法是假设最坏的情况，即堆栈展开过程变得疯狂。相关的问题是接下来会发生什么？申请在没有任何通知的情况下被突然终止。就像在 win32 案例中一样，人们可以争辩说，系统至少可以在事件日志中记录" unwind procedure went berserk in x.exe at address n"，但是没有留下任何失败的痕迹。

Now, when we understand significance of the `UNWIND_INFO` structure, let's discuss what's in it and/or how it's processed. First of all it is checked for presence of reference to custom language−specific exception handler. If there is one, then it's invoked. Depending on the return value, execution flow is resumed (exception is said to be "handled"), *or* rest of `UNWIND_INFO` structure is processed as following. Beside optional reference to custom handler, it carries information about current callee's stack frame and where non−volatile registers are saved. Information is detailed enough to be able to reconstruct contents of caller's non−volatile registers upon call to current callee. And so caller's context is reconstructed, and then unwind procedure is repeated, i.e. another `UNWIND_INFO` structure is associated, this time, with caller's instruction pointer, which is then checked for presence of reference to language−specific handler, etc. The procedure is recursively repeated till exception is handled. As last resort system "handles" it by generating memory core dump and terminating the application.
现在，当我们理解了 UNWIND _ info 结构的重要性之后，让我们讨论一下它包含了什么内容以及/或者它是如何被处理的。首先检查它是否存在对特定语言异常处理程序的引用。如果有一个，那么它被调用。根据返回值，执行流将恢复(异常称为"处理")，或者 UNWIND _ info 结构的其余部分将按以下方式处理。除了对自定义处理程序的可选引用外，它还携带有关当前被调用方的堆栈帧和非易失性寄存器保存位置的信息。这些信息足够详细，可以在调用当前被调用方时重建调用方非易失性寄存器的内容。因此，调用者的上下文被重构，然后展开过程被重复，即另一个 UNWIND _ info 结构与调用者的指令指针相关联，然后检查是否存在对特定语言处理程序的引用等。这个过程被递归地重复，直到异常被处理。作为最后的手段，系统通过生成内存核心转储和终止应用程序来"处理"它。

As for the moment of this writing NASM unfortunately does not facilitate generation of above mentioned detailed information about stack frame layout. But as of version 2.03 it implements building blocks for generating structures involved in stack unwinding. As simplest example, here is how to deploy custom exception handler for leaf function:
不幸的是，在写这篇文章的时候，NASM 没有提供上面提到的关于堆栈框架布局的详细信息。但是在 2.03 版本中，它实现了构建块来生成涉及到栈展开的结构。作为一个简单的例子，下面是如何为 leaf 函数部署自定义异常处理程序:

```
default
违约      rel
section   .text
部分      短信
```

```
extern      MessageBoxA
外科医生 信息框 a
handler:
处理人:
            sub         rsp,40
            字幕        Rsp，40
            mov         rcx,0
            动起来      Rcx 0
            lea         rdx,[text]
            :           Rdx [文本]
            lea         r8,[caption]
            :           R8，[标题]
            mov         r9,1        ; MB_OKCANCEL
            动起来      R9,1        MB _ okcancel
            call        MessageBoxA
            打电话      信息框 a
            sub         eax,1       ; incidentally suits as return value
            字幕        Eax，1      顺便说一下，适合作为回报价值
                                    ; for exception handler
                                    ; 对于异常处理程序
            add         rsp,40
            添加        Rsp，40
            ret
            后悔
global      main
全球        主要
main:
主要:
                        rax,rax
            xor         雷克斯，
            Xor         雷克斯
            mov         rax,QWORD[rax]  ; cause exception
            动起来      引起例外
            ret
            后悔
main_end:
主要结尾:
text:       db          ’OK to rethrow, CANCEL to generate core dump’,0
文字:       数据库      ’可以重新抛出，取消生成核心转储’，0
                        ’SEGV’,0
caption:db              “ SEGV”，
标题: 数据库            0
```

93

```
        .pdata
section 个人资   rdata align=4
部分    料       Rdata align = 4
        dd
        编译:
        pestwav  main wrt ..imagebase
        e        主文字，图像库
        dd
        编译:
        pestwav  main_end wrt ..imagebase
        e        图像库
        dd
        编译:
        pestwav  xmain wrt ..imagebase
        e        图片库
section .xdata   rdata align=8
部分    Xdata    Rdata align = 8
xmain:  db       9,0,0,0
Xmain:  数据库   9,0,0,0
        ad       handler
wrt ..imagebase section .drectve info
        处理程序 wrt。图像库部分
        db       '/defaultlib:user32.lib /defaultlib:msvcrt.lib '
        Db'/defaultlib: user32.lib/defaultlib: msvcrt.lib'
```

What you see in `.pdata` section is element of the "table comprising start and end addresses of function" along with reference to associated `UNWIND_INFO` structure. And what you see in `.xdata` section is `UNWIND_INFO` structure describing function with no frame, but with designated exception handler. References are *required* to be image−relative (which is the real reason for implementing `wrt ..imagebase` operator). It should be noted that `rdata align=n`, as well as `wrt ..imagebase`, are optional in these two segments' contexts, i.e. can be omitted. Latter means that *all* 32−bit references, not only above listed required ones, placed into these two segments turn out image−relative. Why is it important to understand? Developer is allowed to append handler−specific data to `UNWIND_INFO` structure, and if [s]he adds a 32−bit reference, then [s]he will have to remember to adjust its value to obtain the real pointer.
你看到了什么。Pdata 部分是"包含函数起始地址和结束地址的表"的元素，以及相关的 UNWIND _ info 结构。以及你在。Xdata 部分是 UNWIND _ info 结构，描述的函数没有框架，但有指定的异常处理程序。引用必须是图像相关的(这是实现 wrt 的真正原因)。.Imagebase 操作符)。值得注意的是，rdata align = n，以及 wrt。.Imagebase，在这两个片段的上下文中是可选的，也就是说可以省略。后者意味着所有的 32 位引用，不仅仅是上面列出的必需的引用，放置在这两个片段中，结果是图像相对的。为什么理解它很重要？开发人员被允许在 UNWIND _ info 结构中附加特定于处理程序的数据，如果他添加了一个 32 位的引用，那么他必须记住调整它的值以获得真正的指针。

As already mentioned, in Win64 terms leaf function is one that does not call any other function *nor* modifies any non−volatile register, including stack pointer. But it's not uncommon that assembler programmer plans to utilize every single register and sometimes even have variable stack frame. Is there anything one can do with bare building blocks? I.e. besides manually composing fully−fledged `UNWIND_INFO` structure, which would surely be considered error−prone? Yes, there is. Recall that exception handler is called first, before stack layout is analyzed. As it turned out, it's perfectly possible to manipulate current callee's context in custom handler in manner that permits further stack unwinding. General idea is that handler would not actually "handle" the exception, but instead restore callee's context, as it was at its entry point and thus mimic leaf function. In other words, handler would simply undertake part of unwinding procedure. Consider following example:
如前所述，在 win64 术语中，leaf 函数不调用任何其他函数，也不修改任何非易失性寄存器，包括堆栈指针。但是汇编程序员计划使用每一个寄存器，有时甚至有可变的堆栈帧，这种情况并不少见。有没有什么方法可以使用单独的构建块呢？例如，除了手工构建成熟的 UNWIND _ info 结构，哪个结构肯定会

被认为是容易出错的？是的，有。回想一下，在分析堆栈布局之前，先调用异常处理程序。事实证明，完全可以在自定义处理程序中操作当前被调用方的上下文，以允许进一步的堆栈展开。一般的想法是，handler 实际上不会“处理”异常，而是恢复被调用方的上下文，就像它在入口点时那样，从而模拟 leaf 函数。换句话说，handler 只是承担了展开过程的一部分。考虑下面的例子:

```
function:
功能:
          mov      rax,rsp                copy rsp to volatile register
          动起来   雷克斯，rsp            ; 将 rsp 复制到 volatile 寄存器
          push     r15                    save non−volatile registers
          推       R15                    ; 保存非易失性寄存器
          push     rbx
          推       Rbx
          push     rbp
          推       Rbp
          mov      r11,rsp                prepare variable stack frame
          动起来   R11，rsp               ; 准备可变堆栈框架
          sub      r11,rcx
          字幕     R11，rcx
          and      r11,−64
          及       R11,-64
          mov      QWORD[r11],rax         check for exceptions
          动起来   QWORD [ r11] rax       ; 检查例外情况
          mov      rsp,r11                allocate stack frame
          动起来   Rsp，r11               ; 分配堆栈帧
          mov      QWORD[rsp],rax         save original rsp value
          动起来   雷克斯，雷克斯         ; 保存原始 rsp 值
magic_point:
魔法点:
          ...
          ...
          mov      r11,QWORD[rsp]         pull original rsp value
          动起来   R11，QWORD [ rsp ]     ; 提取原始 rsp 值
                   rbp,QWORD[r11−24]
          mov      Rbp，QWORD [ r11-
          动起来   24]
          mov      rbx,QWORD[r11−16]
          动起来   R11-16
                   r15,QWORD[r11−8]
          mov      R15，QWORD [ r11-
          动起来   8]
          mov      rsp,r11                destroy frame
          动起来   Rsp，r11               ; 摧毁框架
          ret
          后悔
```

The keyword is that up to `magic_point` original `rsp` value remains in chosen volatile register and no non−volatile register, except for `rsp`, is modified. While past `magic_point rsp` remains constant till the very end of the `function`. In this case custom language−specific exception handler would look like this:
关键字是，直到魔术 _ 点原始的 rsp 值保留在选择的挥发寄存器和没有非挥发寄存器，除了 rsp，被修改。而过去的 magic _ point rsp 直到函数结束都保持不变。在这种情况下，自定义语言特定的异常处理程序看起来像这样:

94

```
EXCEPTION_DISPOSITION handler (EXCEPTION_RECORD *rec,ULONG64 frame,
        CONTEXT *context,DISPATCHER_CONTEXT *disp)
处理程序(EXCEPTION _ record * rec，ulong64 帧，CONTEXT * CONTEXT，
        dispatcher _ CONTEXT * disp)
{   ULONG64 *rsp;
{ ULONG64 * rsp;
    if (context->Rip<(ULONG64)magic_point)
    如果(上下文-> Rip < (ULONG64)魔法 _ 点)
        rsp = (ULONG64 *)context->Rax;
        Rsp = (ULONG64 *) context-> Rax;
    else
    其他
    { rsp = ((ULONG64 **)context->Rsp)[0];
        context->Rbp = rsp[-3];
        context->Rbx = rsp[-2];
        context->R15 = rsp[-1];
    { Rsp = ((ULONG64 * *) context->
        Rsp)[0] ; context-> Rbp = Rsp [-3] ;
        context-> Rbx = Rsp [-2] ; context-
        > R15 = Rsp [-1] ;
    }
    context->Rsp = (ULONG64)rsp;
    上下文-> Rsp = (ULONG64) Rsp;

    memcpy (disp->ContextRecord,context,sizeof(CONTEXT));
    RtlVirtualUnwind(UNW_FLAG_NHANDLER,disp->ImageBase,
    RtlVirtualUnwind (UNW _ flag _ nhandler，disp->
    ImageBase,
        dips->ControlPc,disp->FunctionEntry,disp->ContextRecord,
        dips-> ControlPc，disp-> FunctionEntry，disp-> ContextRecord,
        &disp->HandlerData,&disp->EstablisherFrame,NULL);
    return ExceptionContinueSearch;
        & disp-> HandlerData，& disp-> EstablisherFrame，
    NULL) ; 返回异常;
}
```

As custom handler mimics leaf function, corresponding UNWIND_INFO structure does not have to contain any information about stack frame and its layout.
作为自定义处理器模拟叶函数，相应的 UNWIND _ info 结构不必包含任何关于堆栈框架及其布局的信息。

7.7 coff: Common Object File Format

7.7 coff: 通用对象文件格式

The coff output type produces COFF object files suitable for linking with the DJGPP linker.
COFF 输出类型产生适合与 DJGPP 链接器链接的 COFF 目标文件。

coff provides a default output file−name extension of .o.
Coff 提供了一个默认的输出文件扩展名.o。

The coff format supports the same extensions to the SECTION directive as win32 does, except that the align qualifier and the info section type are not supported.
Coff 格式和 win32 一样支持 SECTION 指令的扩展，只是不支持 align 修饰符和 info SECTION 类型。

7.8 macho32 and macho64: Mach Object File Format

7.8 macho32 和 macho64: Mach 对象文件格式

The `macho32` and `macho64` output formts produces Mach−O object files suitable for linking with the MacOS X linker. `macho` is a synonym for `macho32`.
Macho32 和 macho64 输出格式产生的 Mach-o 目标文件适合与 MacOS x 链接器链接。Macho 是 macho32 的同义词。

`macho` provides a default output file−name extension of `.o`.
Macho 提供默认的输出文件扩展名.o。

7.8.1 `macho` extensions to the `SECTION` Directive

7.8.1 macho 扩展到 SECTION Directive

The `macho` output format specifies section names in the format "*segment*`,`*section*". No spaces are allowed around the comma. The following flags can also be specified:
Macho 输出格式以"段，段"的格式指定节名。逗号周围不允许有空格。也可以指定以下标志:

• `data` – this section contains initialized data items
数据-这一部分包含初始化的数据项

• `code` – this section contains code exclusively
代码-本节仅包含代码

• `mixed` – this section contains both code and data
混合-本节包含代码和数据

• `bss` – this section is uninitialized and filled with zero
Bss —此部分未初始化并填充为零

• `zerofill` – same as `bss`
Zeroffill-与 bss 相同

• `no_dead_strip` – inhibit dead code stripping for this section
这个部分没有死代码剥离

• `live_support` – set the live support flag for this section
Live _ support —为这个部分设置 live support 标志

• `strip_static_syms` – strip static symbols for this section
带状静态符号-本节的带状静态符号

• `debug` – this section contains debugging information
Debug -- 这一部分包含调试信息

• `align=`*alignment* – specify section alignment
对齐 = 对齐-指定节对齐

The default is `data`, unless the section name is `__text` or `__bss` in which case the default is `text` or `bss`, respectively.
默认值为 data，除非节名为 _ _ text 或 _ _ bss，在这种情况下，默认值分别为 text 或 bss。

For compatibility with other Unix platforms, the following standard names are also supported:
为了与其他 Unix 平台兼容，还支持以下标准名称:

.text	= __TEXT,__text	text
短信	= _ _ TEXT， _ _ TEXT	短信
	= __DATA,__const	
.rodata	= _ _ DATA， _ _	data
. rodata	const	资料
	__DATA,__data	
.data	_ _ DATA， _ _	data
。数据	= DATA	资料
.bss	__DATA,__bss	bss
老板	= _ _ DATA， _ _ bss	Bss

If the `.rodata` section contains no relocations, it is instead put into the `__TEXT,__const` section unless this section has already been specified explicitly. However, it is probably better to specify `__TEXT,__const` and `__DATA,__const` explicitly as appropriate.
如果。Rodata 部分不包含重定位，除非该部分已经被明确指定，否则它将被放入 _ _ TEXT，_ _ const 部分。然而，在适当的情况下显式地指定 _ _ TEXT，_ _ const 和 _ _ DATA，_ const 可能会更好。

7.8.2 Thread Local Storage in Mach−O: `macho` special symbols and `WRT`
7.8.2 线程本地存储 Mach-o: macho 特殊符号和 WRT

Mach−O defines the following special symbols that can be used on the right−hand side of the `WRT` operator:
Mach-o 定义了下列可用于 WRT 操作符右侧的特殊符号:

- `..tlvp` is used to specify access to thread−local storage.

`Tlvp` 用于指定对线程本地存储的访问。

- `..gotpcrel` is used to specify references to the Global Offset Table. The GOT is supported in the `macho64` format only.

`Gotpcrel` 用于指定对全局偏移量表的引用。 `GOT` 只支持 `macho64` 格式。

7.8.3 `macho` specfic directive `subsections_via_symbols`
7.8.3 maco 特定指令子节 _ 通过 _ 符号

The directive `subsections_via_symbols` sets the `MH_SUBSECTIONS_VIA_SYMBOLS` flag in the Mach−O header, that effectively separates a block (or a subsection) based on a symbol. It is often used for eliminating dead codes by a linker.
指令子节 _ via _ symbols 在 Mach-o 标头中设置 MH _ subsections _ via _ symbols 标志，该标志基于符号有效地分隔块(或子节)。它通常用于通过链接器来消除死代码。

This directive takes no arguments.
这个指令没有参数。

This is a macro implemented as a `%pragma`. It can also be specified in its `%pragma` form, in which case it will not affect non−Mach−O builds of the same source code:
这是一个作为% 杂注实现的宏。它也可以以其% 杂注形式指定，在这种情况下，它不会影响同一源代码的非 Mach-o 构建:

```
%pragma macho subsections_via_symbols
% pragma macho subsections _ via _ symbols
```

7.8.4 `macho` specfic directive `no_dead_strip`
7.8.4 大男子主义的特定指令没有死亡地带

The directive `no_dead_strip` sets the Mach−O `SH_NO_DEAD_STRIP` section flag on the section containing a a specific symbol. This directive takes a list of symbols as its arguments.
指令 no _ dead _ strip 将 Mach-o SH _ no _ dead _ strip 区域标志设置在包含特定符号的区域上。这个指令使用一个符号列表作为参数。

This is a macro implemented as a `%pragma`. It can also be specified in its `%pragma` form, in which case it will not affect non−Mach−O builds of the same source code:
这是一个作为% 杂注实现的宏。它也可以以其% 杂注形式指定，在这种情况下，它不会影响同一源代码的非 Mach-o 构建:

```
%pragma macho no_dead_strip symbol...
% pragma macho no _ dead _ strip 符号..。
```

7.8.5 `macho` specific extensions to the `GLOBAL` Directive: `private_extern`
7.8.5 对 GLOBAL Directive: private _ extern 的大男子主义特定扩展

The directive extension to `GLOBAL` marks the symbol with limited global scope. For example, you can specify the global symbol with this extension:
GLOBAL 的指令扩展标记了全局范围有限的符号。例如，你可以用这个扩展名指定全局符号:

```
global foo:private_extern
Global foo: private _ extern
foo:
Foo:
         ; codes
         密码
```

Using with static linker will clear the private extern attribute. But linker option like `-keep_private_externs` can avoid it.
使用静态链接器将清除私有外部属性。但链接器选项，如-keep _ private _ externs 可以避免这种情况。

7.9 `elf32`, `elf64`, `elfx32`: Executable and Linkable Format Object Files
7.9 elf32，elf64，elfx32: 可执行与可链接格式对象文件

The `elf32`, `elf64` and `elfx32` output formats generate `ELF32 and ELF64` (Executable and Linkable Format) object files, as used by Linux as well as Unix System V, including Solaris x86, UnixWare and SCO Unix. `elf` provides a default output file−name extension of `.o`. `elf` is a synonym for `elf32`.
ELF32、 elf64 和 elfx32 输出格式生成 elf32 和 ELF64(可执行与可链接格式)目标文件，正如 Linux 和 Unix System v (包括 Solaris x86、 Unix ware 和 SCO Unix)所使用的那样。Elf 提供了一个默认的输出文件扩展名。Elf 是 elf32 的同义词。

The `elfx32` format is used for the x32 ABI, which is a 32−bit ABI with the CPU in 64−bit mode.
Elfx32 格式用于 x32 ABI，这是一个 32 位 ABI，CPU 为 64 位模式。

7.9.1 ELF specific directive `osabi`
7.9.1 ELF 特定指令

The ELF header specifies the application binary interface for the target operating system (OSABI). This field can be set by using the `osabi` directive with the numeric value (0−255) of the target system. If this directive is not used, the default value will be "UNIX System V ABI" (0) which will work on most systems which support ELF.
ELF 头指定目标操作系统(OSABI)的应用二进制接口。这个字段可以通过使用 osabi 指令和目标系统的数值(0-255)来设置。如果不使用此指令，默认值将是“ UNIX System v ABI”(0)， 它将在大多数支持 ELF 的系统上工作。

7.9.2 `elf` extensions to the `SECTION` Directive
7.9.2 对 SECTION 指令的精灵扩展

Like the `obj` format, `elf` allows you to specify additional information on the `SECTION` directive line, to control the type and properties of sections you declare. Section types and properties are generated automatically by NASM for the standard section names, but may still be overridden by these qualifiers.
像 obj 格式一样，elf 允许你在 SECTION 指令行上指定额外的信息，来控制你声明的章节的类型和属性。Section 类型和属性是由 NASM 为标准的 Section 名称自动生成的，但仍然可能被这些限定符覆盖。

The available qualifiers are:
可用的限定词是:

- `alloc` defines the section to be one which is loaded into memory when the program is run. `noalloc` defines it to be one which is not, such as an informational or comment section.

`Alloc` 定义了一个在程序运行时加载到内存中的区域。`Noalloc` 将其定义为一个不存在的部分，如信息部分或注释部分。

- `exec` defines the section to be one which should have execute permission when the program is run. `noexec` defines it as one which should not.

`Exec` 将节定义为在程序运行时应具有执行权限的节。`Noexec` 将其定义为不应该的。

- `write` defines the section to be one which should be writable when the program is run. `nowrite` defines it as one which should not.

`Write` 定义了程序运行时应该可写的部分。`Nowrite` 将其定义为不应该定义的。

- `progbits` defines the section to be one with explicit contents stored in the object file: an ordinary code or data section, for example, `nobits` defines the section to be one with no explicit contents given, such as a BSS section.

`Progbits` 将节定义为具有存储在对象文件中的显式内容的节： 例如， 普通代码或数据节 `nobits` 将节定义为没有给出显式内容的节，例如 `BSS` 节。

- `align=`, used with a trailing number as in `obj`, gives the alignment requirements of the section.

`Align =` ，与 `obj` 中的尾随数字一起使用，给出了该节的对齐要求。

- `ent=` or `entsize=` specifies the fundamental data item size for a section which contains either fixed−sized data structures or strings; this is generally used with the `merge` attribute (see below.)

Ent = 或 entsize = 指定包含固定大小数据结构或字符串的部分的基本数据项大小；这通常与 merge 属性一起使用(参见下文)

- byte, word, dword, qword, tword, oword, yword, or zword are both shorthand for entsize=, but also sets the default alignment.

Byte，word，dword，qword，tword，oword，yword，或 zword 都是 entsize = 的简写，但也设置了默认的对齐方式。

- strings, ELF attributestrings indicate that this section contains exclusively null−terminated strings. By default these are assumed to be byte strings, but a size specifier can be used to override that.

字符串，ELF 属性字符串表示此部分仅包含以空结尾的字符串。默认情况下，这些字符串被假定为字节字符串，但是可以使用大小说明来覆盖它们。

- merge indicates that duplicate data elements in this section should be merged with data elements from other object files. Data elements can be either fixed−sized objects or null−terminatedstrings (with the strings attribute.) A size specifier is required unless strings is specified, in which case the size defaults to byte.

Merge 指示本节中的重复数据元素应该与其他对象文件中的数据元素合并。数据元素可以是固定大小的对象，也可以是空终止的字符串(带有字符串属性)除非指定了字符串，否则需要一个大小说明符，在这种情况下，大小默认为字节。

- tls defines the section to be one which contains thread local variables.

Tls 将节定义为包含线程局部变量的节。

The defaults assumed by NASM if you do not specify the above qualifiers are:
如果你没有指定上面的限定符，NASM 假设的默认值是:

section .text		progbits	alloc	exec	nowrite	align=16
文本		先知	Alloc	Exec	不要写信	对齐 = 16
section .rodata		progbits	alloc	noexec	nowrite	align=4
罗达数据		先知	Alloc	Noexec	不要写信	对齐 = 4
section	.lrodata	progbits	alloc	noexec	nowrite	align=4
部分	Lrodata	progbits	Alloc	Noexec	不要写信	对齐 = 4
section	.data	progbits	alloc	noexec	write	align=4
部分	。数据	先知	Alloc	Noexec	写作	对齐 = 4

section .ldata	progbits	alloc	noexec	write	align=4	
资料部分	先知	Alloc	Noexec	写作	对齐 = 4	
section .bss	nobits	alloc	noexec	write	align=4	
部门 bss	诺比特	Alloc	Noexec	写作	对齐 = 4	
section .lbss	nobits	alloc	noexec	write	align=4	
Lbss 部门	诺比特	Alloc	Noexec	写作	对齐 = 4	
section .tdata	progbits	alloc	noexec	write	align=4	tls
数据	先知	Alloc	Noexec	写作	对齐 = 4	译注:
section .tbss	nobits	alloc	noexec	write	align=4	tls
部分	诺比特	Alloc	Noexec	写作	对齐 = 4	译注:
section .comment	progbits	noalloc noexec		nowrite	align=1	
评论部分	先知	Noalloc noexec		不要写信	对齐 = 1	
section other	progbits	alloc	noexec	nowrite	align=1	
其他部分	先知	Alloc	Noexec	不要写信	对齐 = 1	

(Any section name other than those in the above table is treated by default like `other` in the above table. Please note that section names are case sensitive.)
(默认情况下，除了上表中的部分名称外，其他任何部分名称都会像上表中的其他部分一样处理。请注意，部分名称是区分大小写的

7.9.3 Position−Independent Code: `macho` Special Symbols and `WRT`
7.9.3 位置无关代码: 大男子主义特殊符号和 WRT

Since `ELF` does not support segment−base references, the `WRT` operator is not used for its normal purpose; therefore NASM's `elf` output format makes use of `WRT` for a different purpose, namely the PIC−specific relocation types.
由于 ELF 不支持基于段的引用，WRT 操作符不用于其正常用途; 因此，NASM 的 ELF 输出格式将 WRT 用于不同的用途，即特定于 PIC 的重定位类型。

`elf` defines five special symbols which you can use as the right−hand side of the `WRT` operator to obtain PIC relocation types. They are `..gotpc`, `..gotoff`, `..got`, `..plt` and `..sym`. Their functions are summarized here:
Elf 定义了五个特殊符号，您可以使用它们作为 WRT 操作符的右侧来获得 PIC 重定位类型。它们是。.Gottpc..下车。.走了。.和。.系统。它们的功能总结如下:

• Referring to the symbol marking the global offset table base using `wrt ..gotpc` will end up giving the distance from the beginning of the current section to the global offset table. (`_GLOBAL_OFFSET_TABLE_` is the standard symbol name used to refer to the GOT.) So you would then need to add `$$` to the result to get the real address of the GOT.
参考使用 wrt 标记全局偏移量表基的符号。.Gotpc 最终会给出从当前部分开始到全局偏移量表的距离。(_ GLOBAL _ offset _ table _ 是用来指代 GOT 的标准符号名称所以你需要在结果中添加$$来得到 GOT 的真实地址。

• Referring to a location in one of your own sections using `wrt ..gotoff` will give the distance from the beginning of the GOT to the specified location, so that adding on the address of the GOT would give the real address of the location you wanted.
使用 wrt 引用您自己部分中的某个位置。.Gotoff 将给出从 GOT 开始到指定位置的距离，因此在 GOT 的地址上加上这个地址就会得到您想要的位置的真实地址。

• Referring to an external or global symbol using `wrt ..got` causes the linker to build an entry *in* the GOT containing the address of the symbol, and the reference gives the distance from the beginning of the GOT to the entry; so you can add on the address of the GOT, load from the resulting address, and end up with the address of the symbol.
使用 wrt 引用一个外部或全局符号。.GOT 导致链接器在 GOT 中构建一个包含符号地址的条目，引用给出了从 GOT 开始到条目的距离，因此可以添加 GOT 的地址，从结果地址加载，最终得到符号的地址。

- Referring to a procedure name using `wrt ..plt` causes the linker to build a procedure linkage table entry for the symbol, and the reference gives the address of the PLT entry. You can only use this in contexts which would generate a PC−relative relocation normally (i.e. as the destination for `CALL` or `JMP`), since ELF contains no relocation type to refer to PLT entries absolutely.

使用 wrt 引用过程名。.PLT 使链接器为符号构建一个过程链接表条目，引用给出了 PLT 条目的地址。您只能在正常情况下生成 PC 相对重定位的上下文中使用它(即作为 CALL 或 JMP 的目标)，因为 ELF 不包含绝对引用 PLT 条目的重定位类型。

- Referring to a symbol name using `wrt ..sym` causes NASM to write an ordinary relocation, but instead of making the relocation relative to the start of the section and then adding on the offset to the symbol, it will write a relocation record aimed directly at the symbol in question. The distinction is a necessary one due to a peculiarity of the dynamic linker.

使用 wrt 引用符号名。.Sym 会导致 NASM 写入一个普通的重定位，但是它将写入一个直接针对该符号的重定位记录，而不是使重定位相对于该节的开始，然后在该符号上添加偏移量。由于动态链接器的特殊性，这种区别是必要的。

A fuller explanation of how to use these relocation types to write shared libraries entirely in NASM is given in section 9.2.
第 9.2 节对如何使用这些重定位类型完全在 NASM 中编写共享库给出了更全面的解释。

7.9.4 Thread Local Storage in ELF: `elf` Special Symbols and `WRT`
7.9.4 ELF 中的线程本地存储: ELF 特殊符号和 WRT

- In ELF32 mode, referring to an external or global symbol using `wrt ..tlsie` causes the linker to build an entry *in* the GOT containing the offset of the symbol within the TLS block, so you can access the value of the symbol with code such as:

在 elf32 模式中，使用 wrt 引用外部或全局符号。.Tlsie 使链接器在 GOT 中构建一个条目，其中包含 TLS 块中符号的偏移量，因此您可以使用以下代码访问符号的值:

```
mov  eax,[tid wrt ..tlsie]
Mov eax，[ tid wrt. tlsie ]
mov  [gs:eax],ebx
电影，电影
```

- In ELF64 or ELFx32 mode, referring to an external or global symbol using `wrt ..gottpoff` causes the linker to build an entry *in* the GOT containing the offset of the symbol within the TLS block, so you can access the value of the symbol with code such as:

在 elf64 或 elfx32 模式下，使用 wrt 引用外部或全局符号。.Gottpoff 导致链接器在 GOT 中构建一个条目，其中包含 TLS 块中符号的偏移量，因此您可以使用以下代码访问符号的值:

```
        mov   rax,[rel tid wrt ..gottpoff]
        Mov rax，[ rel tid wrt。 gottoff ]
        mov   rcx,[fs:rax]
        Mov rcx，[ fs: rax ]
```

7.9.5 `elf` Extensions to the `GLOBAL` Directive
7.9.5 对 GLOBAL 指令的精灵扩展

`ELF` object files can contain more information about a global symbol than just its address: they can contain the size of the symbol and its type as well. These are not merely debugger conveniences, but are actually necessary when the program being written is a shared library. NASM therefore supports some extensions to the `GLOBAL` directive, allowing you to specify these features.
ELF 对象文件可以包含有关全局符号的更多信息，而不仅仅是其地址：它们还可以包含符号的大小和类型。这些不仅仅是调试器的便利，而且在被编写的程序是一个共享库的情况下实际上是必要的。因此，NASM 支持对 GLOBAL 指令的一些扩展，允许你指定这些特性。

You can specify whether a global variable is a function or a data object by suffixing the name with a colon and the word `function` or `data`. (`object` is a synonym for `data`.) For example:
你可以指定一个全局变量是一个函数还是一个数据对象，方法是在名称后面加上冒号和单词 function 或 data。(object 是数据的同义词)例如:

```
global   hashlookup:function, hashtable:data
全局 hashlookup: function，hashtable: data
```

exports the global symbol `hashlookup` as a function and `hashtable` as a data object.
将全局符号 hashlookup 导出为函数，hashtable 导出为数据对象。

Optionally, you can control the ELF visibility of the symbol. Just add one of the visibility keywords: `default`, `internal`, `hidden`, or `protected`. The default is `default` of course. For example, to make `hashlookup` hidden:
您可以选择控制符号的 ELF 可见性。只需添加一个可见性关键字: 默认、内部、隐藏或保护。默认值是默认值。例如，隐藏 hashlookup:

```
global   hashlookup:function hidden
全局 hashlookup: 函数隐藏
```

You can also specify the size of the data associated with the symbol, as a numeric expression (which may involve labels, and even forward references) after the type specifier. Like this:
您还可以在类型说明符后面指定与符号关联的数据的大小，作为数值表达式(可能涉及标签，甚至前向引用)。像这样:

```
global  hashtable:data (hashtable.end - hashtable)
全局 hashtable: data (hashtable.end-hashtable)

hashtable:
Hashtable:
        db this,that,theother  ; some data here
        Db 这个，那个，另一个; 这里有一些数据
.end:
结束:
```

This makes NASM automatically calculate the length of the table and place that information into the `ELF` symbol table.
这使得 NASM 自动计算表的长度，并将信息放入 ELF 符号表中。

Declaring the type and size of global symbols is necessary when writing shared library code. For more information, see section 9.2.4.
在编写共享库代码时，必须声明全局符号的类型和大小。欲了解更多信息，请参阅 9.2.4 节。

7.9.6 `elf` Extensions to the `COMMON` Directive
7.9.6 通用指令的精灵扩展

`ELF` also allows you to specify alignment requirements on common variables. This is done by putting a number (which must be a power of two) after the name and size of the common variable, separated (as usual) by a colon. For example, an array of doublewords would benefit from 4−byte alignment:
`ELF` 还允许您指定公共变量的对齐要求。这是通过在公共变量的名称和大小后面放一个数字(必须是 2 的幂)来实现的，通常用冒号分隔。例如，一个双词数组将受益于 4 字节对齐：

```
common  dwordarray 128:4
通用 dwordarray 128:4
```

This declares the total size of the array to be 128 bytes, and requires that it be aligned on a 4−byte boundary.
这将声明数组的总大小为 128 字节，并要求它在 4 字节边界上对齐。

7.9.7 16−bit code and ELF
7.9.716 位代码和 ELF

The `ELF32` specification doesn't provide relocations for 8− and 16−bit values, but the GNU `ld` linker adds these as an extension. NASM can generate GNU−compatible relocations, to allow 16−bit code to be linked as ELF using GNU `ld`. If NASM is used with the `−w+gnu−elf−extensions` option, a warning is issued when one of these relocations is generated.
Elf32 规范不提供 8 位和 16 位值的重定位，但 GNU ld 链接器将这些作为扩展添加。NASM 可以生成与 GNU 兼容的重定位，允许 16 位代码使用 GNU id 作为 ELF 链接。如果 NASM 与 -w + gnu-elf-extensions 选项一起使用，当其中一个重定位生成时会发出警告。

7.9.8 Debug formats and ELF
7.9.8 调试格式和 ELF

ELF provides debug information in `STABS` and `DWARF` formats. Line number information is generated for all executable sections, but please note that only the ".text" section is executable by default.
ELF 以 STABS 和 DWARF 格式提供调试信息。行号信息是为所有可执行部分生成的，但是请注意只有。默认情况下是可执行的。

7.10 `aout`: Linux `a.out` Object Files
7.10 关于: Linux a.out Object Files

The `aout` format generates `a.out` object files, in the form used by early Linux systems (current Linux systems use ELF, see section 7.9.) These differ from other `a.out` object files in that the magic number in the first four bytes of the file is different; also, some implementations of `a.out`, for example NetBSD's, support position−independent code, which Linux's implementation does not.
Aout 格式以早期 Linux 系统使用的格式生成 a.out 对象文件(当前 Linux 系统使用 ELF，参见第 7.9 节) 它们与其他 a.out 对象文件的不同之处在于，文件的前四个字节中的魔法数是不同的; 此外，a.out 的一些实现，例如 NetBSD 的，支持位置无关代码，而 Linux 的实现不支持这种代码。

`a.out` provides a default output file−name extension of `.o`.
Out 提供了默认的输出文件名扩展名.o。

`a.out` is a very simple object format. It supports no special directives, no special symbols, no use of `SEG` or `WRT`, and no extensions to any standard directives. It supports only the three standard section names `.text`, `.data` and `.bss`.
A.out 是一个非常简单的对象格式。它不支持任何特殊的指令，不支持任何特殊的符号，不支持 SEG 或 WRT 的使用，也不支持任何标准指令的扩展。它只支持三个标准的章节名称。文本。资料及。.

7.11 `aoutb`: NetBSD/FreeBSD/OpenBSD `a.out` Object Files
7.11 aoutb: NetBSD/FreeBSD/OpenBSD a.out Object Files

The `aoutb` format generates `a.out` object files, in the form used by the various free `BSD Unix` clones, `NetBSD`, `FreeBSD` and `OpenBSD`. For simple object files, this object format is exactly the same as `aout` except for the magic number in the first four bytes of the file. However, the `aoutb` format supports position−independent code in the same way as the `elf` format, so you can use it to write `BSD` shared libraries.
Aoutb 格式以各种免费的 BSD Unix 克隆、 NetBSD、 FreeBSD 和 OpenBSD 使用的格式生成 a.out 对象文件。对于简单的对象文件，这种对象格式与 aout 完全相同，除了文件的前四个字节中的魔幻数字。然而，aoutb 格式和 elf 格式一样支持位置无关代码，所以你可以用它来编写 BSD 共享库。

`aoutb` provides a default output file−name extension of `.o`.
Aoutb 提供默认的输出文件扩展名.o。

`aoutb` supports no special directives, no special symbols, and only the three standard section names `.text`, `.data` and `.bss`. However, it also supports the same use of `WRT` as `elf` does, to provide position−independent code relocation types. See section 7.9.3 for full documentation of this feature.
Aoutb 不支持特殊的指令，不支持特殊的符号，只支持三个标准的节名。文本。资料及。.然而，它也支持像 elf 一样使用 WRT，提供位置无关的代码重定位类型。关于这个特性的完整文档见第 7.9.3 节。

`aoutb` also supports the same extensions to the `GLOBAL` directive as `elf` does: see section 7.9.5 for documentation of this.
Aoutb 还像 elf 一样支持对 GLOBAL 指令的相同扩展：参见第 7.9.5 节的文档说明。

7.12 `as86`: Minix/Linux `as86` Object Files
7.12 as86: Minix/Linux as86 Object Files

The Minix/Linux 16−bit assembler `as86` has its own non−standard object file format. Although its companion linker `ld86` produces something close to ordinary `a.out` binaries as output, the object file format used to communicate between `as86` and `ld86` is not itself `a.out`.
Minix/Linux 16 位汇编程序 as86 有自己的非标准目标文件格式。虽然它的伙伴链接器 ld86 生成的输出类似于普通的 a.out 二进制文件，但用于在 as86 和 ld86 之间通信的对象文件格式本身并不是 a.out。

NASM supports this format, just in case it is useful, as `as86`. `as86` provides a default output file−name extension of `.o`.
NASM 支持这种格式，以防万一，比如 as86。As86 提供了一个默认的输出文件扩展名。O.

`as86` is a very simple object format (from the NASM user's point of view). It supports no special directives, no use of `SEG` or `WRT`, and no extensions to any standard directives. It supports only the three standard section names `.text`, `.data` and `.bss`. The only special symbol supported is
As86 是一种非常简单的对象格式(从 NASM 用户的角度来看)。它不支持任何特殊的指令，不使用 SEG 或 WRT，也不支持任何标准指令的扩展。它只支持三个标准的章节名称。文本。资料及。.唯一受支持的特殊符号是
`..start`.
开始。

7.13 `rdf`: Relocatable Dynamic Object File Format

7.13 rdf: Relocatable Dynamic Object File Format

The `rdf` output format produces `RDOFF` object files. `RDOFF` (Relocatable Dynamic Object File Format) is a home−grown object−file format, designed alongside NASM itself and reflecting in its file format the internal structure of the assembler.
Rdf 输出格式产生 RDOFF 对象文件。RDOFF (Relocatable Dynamic Object File Format)是一种自主开发的对象文件格式，与 NASM 本身一起设计，并以其文件格式反映汇编程序的内部结构。

`RDOFF` is not used by any well−known operating systems. Those writing their own systems, however, may well wish to use `RDOFF` as their object format, on the grounds that it is designed primarily for simplicity and contains very little file−header bureaucracy.
任何著名的操作系统都不使用 RDOFF。然而，那些编写自己的系统的人很可能希望使用 RDOFF 作为他们的对象格式，理由是它主要是为简单性而设计的，并且几乎不包含文件头官僚主义。

The Unix NASM archive, and the DOS archive which includes sources, both contain an `rdoff` subdirectory holding a set of RDOFF utilities: an RDF linker, an `RDF` static−library manager, an RDF file dump utility, and a program which will load and execute an RDF executable under Linux.
Unix NASM 存档和包含源代码的 DOS 存档都包含一个 RDOFF 子目录，其中包含一组 RDOFF 实用程序: 一个 RDF 链接器、一个 RDF 静态库管理器、一个 RDF 文件转储实用程序，以及一个在 Linux 下加载和执行 RDF 可执行文件的程序。

`rdf` supports only the standard section names `.text`, `.data` and `.bss`.
Rdf 只支持标准的节名.text、 .data 和.bss。

7.13.1 Requiring a Library: The `LIBRARY` Directive
7.13.1 需要图书馆: 图书馆指令

`RDOFF` contains a mechanism for an object file to demand a given library to be linked to the module, either at load time or run time. This is done by the `LIBRARY` directive, which takes one argument which is the name of the module:
`RDOFF` 包含一个目标文件的机制，要求给定的库在加载时或运行时链接到模块。这是由 `LIBRARY` 指令完成的，该指令带有一个参数，即模块的名称：

```
library  mylib.rdl
Library mylib.rdl
```

7.13.2 Specifying a Module Name: The `MODULE` Directive
7.13.2 指定模块名称: 模块指令

Special `RDOFF` header record is used to store the name of the module. It can be used, for example, by run−time loader to perform dynamic linking. `MODULE` directive takes one argument which is the name of current module:
特殊的 RDOFF 头记录用于存储模块的名称。例如，它可以被运行时加载器用来执行动态链接。MODULE 指令接受一个参数，即当前模块的名称:

```
module  mymodname
模块 mymodname
```

Note that when you statically link modules and tell linker to strip the symbols from output file, all module names will be stripped too. To avoid it, you should start module names with `$`, like:
注意，当您静态链接模块并告诉链接器从输出文件中删除符号时，所有模块名称也将被删除。为了避免这种情况，你应该以 $开始模块名，比如:

```
    module      $kernel.core
        模块 $kernel.core
```

7.13.3 `rdf` Extensions to the `GLOBAL` Directive
7.13.3 对 GLOBAL 指令的 rdf 扩展

`RDOFF` global symbols can contain additional information needed by the static linker. You can mark a global symbol as exported, thus telling the linker do not strip it from target executable or library file. Like in `ELF`, you can also specify whether an exported symbol is a procedure (function) or data object.
`RDOFF` 全局符号可以包含静态链接器所需的额外信息。你可以将一个全局符号标记为导出，从而告诉链接器不要将其从目标可执行文件或库文件中删除。像 `ELF` 一样，你也可以指定导出的符号是过程(函数)还是数据对象。

Suffixing the name with a colon and the word `export` you make the symbol exported:
在名称后面加上冒号，然后将符号导出为 export:

```
global  sys_open:export
Global sys _ open: export
```

To specify that exported symbol is a procedure (function), you add the word `proc` or `function` after declaration:
若要指定导出的符号是一个过程(函数)，请在声明后添加单词 proc 或函数:

```
global  sys_open:export proc
Global sys _ open: export proc
```

Similarly, to specify exported data object, add the word `data` or `object` to the directive:
同样，若要指定导出的数据对象，请在指令中添加单词 data 或 object:

```
global  kernel_ticks:export data
Global kernel _ ticks: 导出数据
```

7.13.4 `rdf` Extensions to the `EXTERN` Directive
7.13.4 对 EXTERN 指令的 rdf 扩展

By default the `EXTERN` directive in `RDOFF` declares a "pure external" symbol (i.e. the static linker will complain if such a symbol is not resolved). To declare an "imported" symbol, which must be resolved later during a dynamic linking phase, `RDOFF` offers an additional `import` modifier. As in `GLOBAL`, you can also specify whether an imported symbol is a procedure (function) or data object. For example:
默认情况下，RDOFF 中的 externn 指令声明了一个“纯外部”符号(即如果这样的符号没有解析，静态链接器将会抱怨)。为了声明一个必须在动态链接阶段解析的“导入”符号，RDOFF 提供了一个额外的导入修饰符。在 GLOBAL 中，你也可以指定导入的符号是过程(函数)还是数据对象。例如:

```
library $libc
图书馆 $libc
extern  _open:import
Extern _ open: 导入
extern  _printf:import proc
输入程序
extern  _errno:import data
Extern _ errno: 导入数据
```

Here the directive `LIBRARY` is also included, which gives the dynamic linker a hint as to where to find requested symbols.
这里还包括指令 LIBRARY，它向动态链接器提示在哪里可以找到所请求的符号。

7.14 `dbg`: Debugging Format

7.14 dbg: 调试格式

The `dbg` format does not output an object file as such; instead, it outputs a text file which contains a complete list of all the transactions between the main body of NASM and the output−format back end module. It is primarily intended to aid people who want to write their own output drivers, so that they can get a clearer idea of the various requests the main program makes of the output driver, and in what order they happen.
Dbg 格式并不输出这样的目标文件; 相反，它输出一个文本文件，其中包含 NASM 主体和 output-format 后端模块之间所有事务的完整列表。它主要用于帮助那些想要编写自己的输出驱动程序的人，以便他们能够更清楚地了解主程序对输出驱动程序发出的各种请求，以及它们发生的顺序。

For simple files, one can easily use the dbg format like this:
对于简单的文件，可以像这样轻松地使用 dbg 格式:

```
nasm -f dbg filename.asm
Nasm-f dbg filename.asm
```

which will generate a diagnostic file called filename.dbg. However, this will not work well on files which were designed for a different object format, because each object format defines its own macros (usually user−level forms of directives), and those macros will not be defined in the dbg format. Therefore it can be useful to run NASM twice, in order to do the preprocessing with the native object format selected:
它将生成一个名为 filename.dbg 的诊断文件。但是，对于为不同对象格式设计的文件，这种方法将不能很好地工作，因为每种对象格式都定义了自己的宏(通常是用户级的指令形式)，而且这些宏不会以 dbg 格式定义。因此，运行两次 NASM 对于选择本地对象格式进行预处理是很有用的:

```
nasm -e -f rdf -o rdfprog.i
rdfprog.asm nasm -a -f dbg rdfprog.i
Nasm-e-f rdf-o rdfprog.i rdfprog.asm
nasm-a-f dbg rdfprogi
```

This preprocesses rdfprog.asm into rdfprog.i, keeping the rdf object format selected in order to make sure RDF special directives are converted into primitive form correctly. Then the preprocessed source is fed through the dbg format to generate the final diagnostic output.
这将把 rdfprog.asm 预处理为 rdfprog.i，保持 RDF 对象格式的选择，以确保 RDF 特殊指令被正确地转换为基元格式。然后通过 dbg 格式输入预处理的数据源，生成最终的诊断输出。

This workaround will still typically not work for programs intended for obj format, because the obj SEGMENT and GROUP directives have side effects of defining the segment and group names as symbols; dbg will not do this, so the program will not assemble. You will have to work around that by defining the symbols yourself (using EXTERN, for example) if you really need to get a dbg trace of an obj–specific source file.
这种解决方案通常仍然不适用于用于 obj 格式的程序，因为 obj SEGMENT 和 GROUP 指令有将段名和组名定义为符号的副作用; dbg 不会这样做，所以程序不会汇编。如果您真的需要获得一个特定于对象的源文件的 dbg 跟踪，那么您将不得不自己定义符号(例如使用 EXTERN)来解决这个问题。

dbg accepts any section name and any directives at all, and logs them all to its output file.
Dbg 接受任何节名和任何指令，并将它们全部记录到其输出文件中。

dbg accepts and logs any %pragma, but the specific %pragma:
Dbg 接受并记录任何% pragma，但是特定的% pragma:

```
%pragma dbg maxdump <size>
% pragma dbg maxdump < size >
```

where <size> is either a number or unlimited, can be used to control the maximum size for dumping the full contents of a rawdata output object.
如果 < size > 是一个数字或者无限大，可以用来控制原始数据输出对象的最大大小。

102
102

Chapter 8: Writing 16−bit Code (DOS, Windows 3/3.1)
第八章: 编写 16 位代码(DOS，Windows 3/3.1)

This chapter attempts to cover some of the common issues encountered when writing 16−bit code to run under `MS−DOS` or `Windows 3.x`. It covers how to link programs to produce `.EXE` or `.COM` files, how to write `.SYS` device drivers, and how to interface assembly language code with 16−bit C compilers and with Borland Pascal.
本章试图涵盖在 MS-DOS 或 windows3.x 下编写 16 位代码时遇到的一些常见问题。它涵盖了如何将程序链接到产品。EXE 或。COM 文件，如何编写。SYS 设备驱动程序，以及如何用 16 位 c 编译器和 Borland Pascal 接口汇编语言代码。

8.1 Producing `.EXE` Files
8.1 制作.EXE 文件

Any large program written under DOS needs to be built as a `.EXE` file: only `.EXE` files have the necessary internal structure required to span more than one 64K segment. Windows programs, also, have to be built as `.EXE` files, since Windows does not support the `.COM` format.
任何在 DOS 下编写的大型程序都需要构建为一个。EXE 文件: 只有。EXE 文件具有跨越多个 64k 段所需的必要内部结构。Windows 程序也必须构建为。EXE 文件，因为 Windows 不支持。COM 格式。

In general, you generate `.EXE` files by using the `obj` output format to produce one or more `.OBJ` files, and then linking them together using a linker. However, NASM also supports the direct generation of simple DOS `.EXE` files using the `bin` output format (by using `DB` and `DW` to construct the `.EXE` file header), and a macro package is supplied to do this. Thanks to Yann Guidon for contributing the code for this.
一般来说，生成。通过使用 obj 输出格式生成一个或多个 EXE 文件。OBJ 文件，然后使用链接器将它们连接在一起。然而，NASM 也支持直接生成简单的 DOS。EXE 文件使用 bin 输出格式(通过使用 DB 和 DW 来构造。EXE 文件头)，并且提供了一个宏包来做这件事。感谢 Yann Guidon 为此贡献了代码。

NASM may also support `.EXE` natively as another output format in future releases.
NASM 也可能支持.EXE 作为未来版本中的另一种输出格式。

8.1.1 Using the `obj` Format To Generate `.EXE` Files
8.1.1 使用 obj 格式生成.EXE 文件

This section describes the usual method of generating `.EXE` files by linking `.OBJ` files together.
本节描述通过将.OBJ 文件链接在一起来生成.EXE 文件的常用方法。

Most 16−bit programming language packages come with a suitable linker; if you have none of these, there is a free linker called VAL, available in `LZH` archive format from `x2ftp.oulu.fi`. An LZH archiver can be found at `ftp.simtel.net`. There is another ‘free’ linker (though this one doesn’t come with sources) called FREELINK, available from `www.pcorner.com`. A third, `djlink`, written by DJ Delorie, is available at `www.delorie.com`. A fourth linker, `ALINK`, written by Anthony A.J. Williams, is available at `alink.sourceforge.net`.
大多数 16 位编程语言包都带有一个合适的链接器; 如果您没有这些链接器，那么有一个名为 VAL 的免费链接器，可以从 x2ftp.oulu.fi 获得 lzharchive 格式的链接器。LZH 归档文件可以在 ftp.simtel. net 找到。还有另一个“免费”链接(虽然这个链接没有来源)，叫做 FREELINK，可以从 www.pcorner. com 获得。第三个是 DJ Delorie 写的 djlink，可以在 www.Delorie. com 找到。第四个链接器，由 Anthony a.j。 Williams 写的 ALINK，可以在 ALINK.sourceforge. net 上找到。

When linking several `.OBJ` files into a `.EXE` file, you should ensure that exactly one of them has a start point defined (using the `..start` special symbol defined by the `obj` format: see section 7.4.6). If no module defines a start point, the linker will not know what value to give the entry−point field in the output file header; if more than one defines a start point, the linker will not know *which* value to use.

当连接几个。OBJ 文件到一个。在 EXE 文件中，你应该确保其中一个文件有一个定义好的起点(使用。. 由 obj 格式定义的特殊符号: 参见第 7.4.6 节)。如果没有模块定义起始点，链接器将不知道在输出文件头中给入口点字段赋什么值; 如果有多个模块定义起始点，链接器将不知道使用哪个值。

An example of a NASM source file which can be assembled to a `.OBJ` file and linked on its own to a `.EXE` is given here. It demonstrates the basic principles of defining a stack, initialising the segment registers, and declaring a start point. This file is also provided in the `test` subdirectory of the NASM archives, under the name `objexe.asm`.
一个 NASM 源文件的例子，它可以被组装到一个。OBJ 文件，单独链接到一个。EXE 在这里给出。它演示了定义堆栈、初始化段寄存器和声明起始点的基本原则。这个文件也在 NASM 归档文件的 test 子目录中提供，名为 objexe.asm。

```
segment code
段码

..start:
开始:
        mov     ax,data
        Mov ax 数据
        mov     ds,ax
        Mov ds, ax
        mov     ax,stack
        Mov ax stack
        mov     ss,ax
        移动，斧头
        mov     sp,stacktop
        Mov sp, stacktop
```

This initial piece of code sets up `DS` to point to the data segment, and initializes `SS` and `SP` to point to the top of the provided stack. Notice that interrupts are implicitly disabled for one instruction after a move into `SS`, precisely for this situation, so that there's no chance of an interrupt occurring between the loads of `SS` and `SP` and not having a stack to execute on.
这段初始代码设置 DS 指向数据段，并初始化 SS 和 SP 指向提供的栈顶。请注意，在进入 SS 之后，一条指令隐式禁用了中断，正是在这种情况下，所以在 SS 和 SP 的负载之间不会发生中断，也不会有堆栈可以执行。

Note also that the special symbol `..start` is defined at the beginning of this code, which means that will be the entry point into the resulting executable file.
还要注意，这个特殊的符号。.Start 是在这段代码的开头定义的，这意味着它将是结果可执行文件的入口点。

```
        mov     dx,hello
        Mov dx 你好
        mov     ah,9
        Mov ah，9
        int     0x21
        Int 0x21 整体 0x21
```

The above is the main program: load `DS:DX` with a pointer to the greeting message (`hello` is implicitly relative to the segment `data`, which was loaded into `DS` in the setup code, so the full pointer is valid), and call the DOS print−string function.
上面是主程序: 用一个指向问候消息的指针加载 DS: DX (hello 隐含地相对于段数据，段数据是在设置代码中加载到 DS 中的，所以完整的指针是有效的)，并调用 DOS print-string 函数。

```
        mov     ax,0x4c00
        Mov ax，0 x4c00
        int     0x21
        Int 0x21 整体 0x21
```

This terminates the program using another DOS system call.
这使用另一个 DOS 系统调用终止程序。

```
segment data
段数据

hello:  db      'hello, world', 13, 10, '$'
Hello: db'hello，world'，13,10,'$
```

The data segment contains the string we want to display.
数据段包含我们想要显示的字符串。

```
segment stack stack
片段堆栈，片段堆栈
        resb 64
        Resb 64
stacktop:
堆叠式的:
```

The above code declares a stack segment containing 64 bytes of uninitialized stack space, and points `stacktop` at the top of it. The directive `segment stack stack` defines a segment *called* `stack`, and also of *type* `STACK`. The latter is not necessary to the correct running of the program, but linkers are likely to issue warnings or errors if your program has no segment of type `STACK`.
上面的代码声明了一个包含 64 字节未初始化堆栈空间的堆栈段，并指向堆栈顶端。指令段堆栈定义了一个叫做堆栈的段，类型也是 STACK。后者对于程序的正确运行是不必要的，但是如果程序没有 STACK 类型的段，链接器可能会发出警告或错误。

The above file, when assembled into a `.OBJ` file, will link on its own to a valid `.EXE` file, which when run will print 'hello, world' and then exit.
上述文件，当组合成一个。OBJ 文件，将自己链接到一个有效的。EXE 文件，在运行时会打印" hello，world"然后退出。

8.1.2 Using the `bin` Format To Generate `.EXE` Files
8.1.2 使用 bin 格式生成.EXE 文件

The `.EXE` file format is simple enough that it's possible to build a `.EXE` file by writing a pure−binary program and sticking a 32−byte header on the front. This header is simple enough that it can be generated using `DB` and `DW` commands by NASM itself, so that you can use the `bin` output format to directly generate `.EXE` files.

那个。EXE 文件格式非常简单，可以构建一个。通过编写一个纯二进制程序，并在前面粘贴一个 32 字节的头文件。这个头非常简单，NASM 本身可以使用 DB 和 DW 命令生成它，因此您可以使用 bin 输出格式直接生成。EXE 文件。

Included in the NASM archives, in the `misc` subdirectory, is a file `exebin.mac` of macros. It defines three macros: `EXE_begin`, `EXE_stack` and `EXE_end`.
在 NASM 归档文件中的 misc 子目录中，有一个 exebin.mac 宏文件。它定义了三个宏: EXE _ begin，EXE _ stack 和 EXE _ end。

To produce a `.EXE` file using this method, you should start by using `%include` to load the `exebin.mac` macro package into your source file. You should then issue the `EXE_begin` macro call (which takes no arguments) to generate the file header data. Then write code as normal for the `bin` format – you can use all three standard sections `.text`, `.data` and `.bss`. At the end of the file you should call the `EXE_end` macro (again, no arguments), which defines some symbols to mark section sizes, and these symbols are referred to in the header code generated by `EXE_begin`.
产生一个。使用这个方法，你应该首先使用% include 来加载 exebin.mac 宏包到源文件中。然后你应该发出 EXE _ begin 宏调用(不带参数)来生成文件头数据。然后像平常一样编写 bin 格式的代码——你可以使用所有三个标准部分。文本。资料及。.在文件的末尾，您应该调用 EXE _ end 宏(同样，没有参数)，它定义了一些符号来标记节的大小，这些符号在 EXE _ begin 生成的头代码中被引用。

In this model, the code you end up writing starts at `0x100`, just like a `.COM` file – in fact, if you strip off the 32−byte header from the resulting `.EXE` file, you will have a valid `.COM` program. All the segment bases are the same, so you are limited to a 64K program, again just like a `.COM` file. Note that an `ORG` directive is issued by the `EXE_begin` macro, so you should not explicitly issue one of your own.
在这个模型中，你最终编写的代码从 0x100 开始，就像。COM 文件——事实上，如果你从结果中去掉 32 字节的头部。EXE 文件，你将有一个有效的。COM 程序。所有的段基是相同的，所以你被限制在一个 64k 的程序，就像一个。COM 文件。请注意，ORG 指令是由 EXE _ begin 宏发出的，因此您不应该显式地发出自己的指令。

You can’t directly refer to your segment base value, unfortunately, since this would require a relocation in the header, and things would get a lot more complicated. So you should get your segment base by copying it out of `CS` instead.
不幸的是，您不能直接引用段基值，因为这将需要在标题中重新定位，而且事情会变得复杂得多。所以你应该通过从 CS 中复制段基来获得段基。

104

On entry to your `.EXE` file, `SS:SP` are already set up to point to the top of a 2Kb stack. You can adjust the default stack size of 2Kb by calling the `EXE_stack` macro. For example, to change the stack size of your program to 64 bytes, you would call `EXE_stack 64`.
进入你的。可执行文件，SS: SP 已经设置为指向 2kb 堆栈的顶部。你可以通过调用 EXE _ stack 宏来调整默认的 2kb 堆栈大小。例如，要将程序的堆栈大小改为 64 字节，你可以调用 EXE _ stack 64。

A sample program which generates a `.EXE` file in this way is given in the `test` subdirectory of the NASM archive, as `binexe.asm`.
一个生成一个。EXE 文件以这种方式在 NASM 归档文件的 test 子目录中给出，作为 binexe.asm。

8.2 Producing `.COM` Files
8.2 制作. COM 文件

While large DOS programs must be written as `.EXE` files, small ones are often better written as `.COM` files. `.COM` files are pure binary, and therefore most easily produced using the `bin` output format.
而大型 DOS 程序必须写成。EXE 文件，小文件通常更好地编写为。COM 文件。.COM 文件是纯二进制的，因此最容易使用 bin 输出格式生成。

8.2.1 Using the `bin` Format To Generate `.COM` Files
8.2.1 使用 bin 格式生成.COM 文件

`.COM` files expect to be loaded at offset `100h` into their segment (though the segment may change). Execution then begins at `100h`, i.e. right at the start of the program. So to write a `.COM` program, you would create a source file looking like
.COM 文件希望在偏移 100 小时加载到它们的段中(尽管段可能会改变)。然后在 100 小时开始执行，也就是在程序开始的时候。所以编写一个。COM 程序，你需要创建一个类似于

```
        org 100h
        Org 100h

section .text
文本

start:
开始:
        ; put your code

here section .data

        把你的代码放在这里

        ; put data items

here section .bss

        把数据项放在这里

        ; put uninitialized data here
        把未初始化的数据放在这里
```

The `bin` format puts the `.text` section first in the file, so you can declare data or BSS items before beginning to write code if you want to and the code will still end up at the front of the file where it belongs.
Bin 格式将。文本部分首先在文件中，因此您可以在开始编写代码之前声明数据或 BSS 项，如果您愿意的话，代码仍然会停留在它所属的文件前面。

The BSS (uninitialized data) section does not take up space in the `.COM` file itself: instead, addresses of BSS items are resolved to point at space beyond the end of the file, on the grounds

that this will be free memory when the program is run. Therefore you should not rely on your BSS being initialized to all zeros when you run.
BSS (未初始化的数据)部分不会占用。COM 文件本身: 相反，BSS 项的地址被解析为指向文件末尾以外的空间，理由是在程序运行时这将是空闲内存。因此，你不应该依赖于你的 BSS 在你运行时被初始化为所有的零。

To assemble the above program, you should use a command

line like `nasm myprog.asm -fbin -o myprog.com`

要组装上述程序，您应该使用命令行，如 nasm myprog.asm-

fbin-o myprog. com

The `bin` format would produce a file called `myprog` if no explicit output file name were specified, so you have to override it and give the desired file name.
如果没有指定显式的输出文件名，bin 格式将生成一个名为 myprog 的文件，因此必须重写它并给出所需的文件名。

8.2.2 Using the `obj` Format To Generate `.COM` Files
8.2.2 使用 obj 格式生成.COM 文件

If you are writing a `.COM` program as more than one module, you may wish to assemble several `.OBJ` files and link them together into a `.COM` program. You can do this, provided you have a linker capable of outputting `.COM` files directly (TLINK does this), or alternatively a converter program such as `EXE2BIN` to transform the `.EXE` file output from the linker into a `.COM` file.
如果你正在写一个。COM 程序作为一个以上的模块，你可能希望组装几个。OBJ 文件，并将它们链接到一个。COM 程序。你可以做到这一点，只要你有一个链接器能够输出。COM 文件直接(TLINK 做这个)，或者一个转换程序如 EXE2BIN 来转换。从链接器输出的 EXE 文件转换为。COM 文件。

If you do this, you need to take care of several things:
如果你这样做，你需要注意以下几件事:

- The first object file containing code should start its code segment with a line like `RESB 100h`. This is to ensure that the code begins at offset `100h` relative to the beginning of the code segment, so
第一个包含代码的目标文件应该以 RESB 100h 这样的行开始它的代码段。这是为了确保代码从相对于代码段开头的偏移量 100h 开始，因此

that the linker or converter program does not have to adjust address references within the file when generating the `.COM` file. Other assemblers use an `ORG` directive for this purpose, but `ORG` in NASM is a format−specific directive to the `bin` output format, and does not mean the same thing as it does in MASM−compatible assemblers.
链接器或转换器程序在生成。COM 文件。其他汇编程序为此目的使用 ORG 指令，但是 NASM 中的 ORG 是一个针对 bin 输出格式的特定格式指令，并不意味着与 MASM 兼容汇编程序中的 ORG 相同。

• You don't need to define a stack segment.
您不需要定义堆栈段。

• All your segments should be in the same group, so that every time your code or data references a symbol offset, all offsets are relative to the same segment base. This is because, when a `.COM` file is loaded, all the segment registers contain the same value.
所有段应该在同一组中，以便每次代码或数据引用符号偏移量时，所有偏移量都相对于同一段基。这是因为，当。COM 文件加载时，所有段寄存器都包含相同的值。

8.3 Producing `.SYS` Files
8.3 制作 SYS 文件

MS−DOS device drivers – `.SYS` files – are pure binary files, similar to `.COM` files, except that they start at origin zero rather than `100h`. Therefore, if you are writing a device driver using the `bin` format, you do not need the `ORG` directive, since the default origin for `bin` is zero. Similarly, if you are using `obj`, you do not need the `RESB 100h` at the start of your code segment.
MS-DOS 设备驱动程序 -。SYS 文件-是纯二进制文件，类似于。COM 文件，只不过它们的起始点是 0 而不是 100 小时。因此，如果使用 bin 格式编写设备驱动程序，则不需要 ORG 指令，因为 bin 的默认原点是零。同样的，如果你使用 obj，你也不需要在代码段开始时使用 RESB 100h。

`.SYS` files start with a header structure, containing pointers to the various routines inside the driver which do the work. This structure should be defined at the start of the code segment, even though it is not actually code.
`.SYS` 文件以头部结构开始，包含指向驱动程序中完成工作的各种例程的指针。这个结构应该在代码段的开头定义，即使它实际上不是代码。

For more information on the format of `.SYS` files, and the data which has to go in the header structure, a list of books is given in the Frequently Asked Questions list for the newsgroup `comp.os.msdos.programmer`.
关于。的格式的更多信息。SYS 文件，以及头结构中必须包含的数据，在 newsgroup comp.os.msdos.programmer 的常见问题列表中给出了一个书籍列表。

8.4 Interfacing to 16−bit C Programs
8.4 与 16 位 c 程序的接口

This section covers the basics of writing assembly routines that call, or are called from, C programs. To do this, you would typically write an assembly module as a `.OBJ` file, and link it with your C modules to produce a mixed−language program.
本节介绍了编写调用 c 程序或从 c 程序调用的汇编例程的基本知识。要做到这一点，你通常需要编写一个汇编模块作为。OBJ 文件，然后把它和 c 模块链接起来，生成一个混合语言的程序。

8.4.1 External Symbol Names
8.4.1 外部符号名称

C compilers have the convention that the names of all global symbols (functions or data) they define are formed by prefixing an underscore to the name as it appears in the C program. So, for example, the function a C programmer thinks of as `printf` appears to an assembly language programmer as `_printf`. This means that in your assembly programs, you can define symbols without a leading underscore, and not have to worry about name clashes with C symbols.
C 编译器有一个约定，它们定义的所有全局符号(函数或数据)的名称都是通过在名称前面加上一个下划线来形成的，就像它在 c 程序中出现的那样。例如，c 程序员认为是 printf 的函数在汇编语言

程序员看来就是 _printf。这意味着在你的汇编程序中，你可以在没有下划线的情况下定义符号，而不用担心与 c 符号的名称冲突。

If you find the underscores inconvenient, you can define macros to replace the `GLOBAL` and `EXTERN` directives as follows:
如果您觉得下划线不方便，可以定义宏来替换如下的 GLOBAL 和 EXTERN 指令:

```
%macro  cglobal 1
% 宏 cglobal 1

  global  _%1
  全球 _% 1
  %define %1 _%1
  定义% 1% 1

%endmacro
% endmacro

%macro  cextern 1
% 宏 cextern 1

  extern  _%1
  外部人员 _% 1
  %define %1 _%1
  定义% 1% 1

%endmacro
% endmacro
```

(These forms of the macros only take one argument at a time; a `%rep` construct could solve this.)
(这些形式的宏一次只接受一个参数;% rep 构造可以解决这个问题

If you then declare an external like this:
如果你像这样声明一个外部函数:

```
cextern printf
```
翻译

then the macro will expand it as
然后宏将它展开为

```
extern  _printf
```
外部打印
```
%define printf _printf
```
定义 printf _ printf

Thereafter, you can reference `printf` as if it was a symbol, and the preprocessor will put the leading underscore on where necessary.
此后，可以像引用符号一样引用 printf，预处理器将在必要的地方放置前导下划线。

The `cglobal` macro works similarly. You must use `cglobal` before defining the symbol in question, but you would have had to do that anyway if you used `GLOBAL`.
Cglobal 宏的工作原理是相似的。在定义有问题的符号之前，你必须使用 cglobal，但是如果你使用 GLOBAL，你就必须这样做。

Also see section 2.1.28.
另见第 2.1.28 节。

8.4.2 Memory Models
8.4.2 内存模型

NASM contains no mechanism to support the various C memory models directly; you have to keep track yourself of which one you are writing for. This means you have to keep track of the following things:
NASM 不包含直接支持各种 c 内存模型的机制; 您必须跟踪自己在为哪个 c 内存模型编写代码。这意味着你必须跟踪以下事情:

• In models using a single code segment (tiny, small and compact), functions are near. This means that function pointers, when stored in data segments or pushed on the stack as function arguments, are 16 bits long and contain only an offset field (the `CS` register never changes its value, and always gives the segment part of the full function address), and that functions are called using ordinary near `CALL` instructions and return using `RETN` (which, in NASM, is synonymous with `RET` anyway). This means both that you should write your own routines to return with `RETN`, and that you should call external C routines with near `CALL` instructions.
在使用单一代码段的模型中(微小、小、紧凑)，函数就在附近。这意味着函数指针，当存储在数据段或作为函数参数推送到堆栈上时，长度为 16 位，只包含一个偏移量字段(CS 寄存器从不改变其值，并始终给出完整函数地址的段部分)，并且使用普通的近似 CALL 指令调用函数，使用 RETN 返回(在 NASM 中，RETN 无论如何都是 RET 的同义词)。这意味着你应该写你自己的例程来使用 RETN 返回，你应该调用外部 c 例程使用近似 CALL 指令。

• In models using more than one code segment (medium, large and huge), functions are far. This means that function pointers are 32 bits long (consisting of a 16−bit offset followed by a 16−bit segment), and that functions are called using `CALL FAR` (or `CALL seg:offset`) and return using `RETF`. Again, you should therefore write your own routines to return with `RETF` and use `CALL FAR` to call external routines.
在使用多个代码段(中型、大型和巨型)的模型中，函数很远。这意味着函数指针长度为 32 位(由 16 位偏移量和 16 位段组成)，函数使用 CALL FAR (或 CALL seg: offset)调用并使用 RETF 返回。同样，你应该编写自己的例程来使用 RETF 返回，并使用 CALL FAR 来调用外部例程。

• In models using a single data segment (tiny, small and medium), data pointers are 16 bits long, containing only an offset field (the `DS` register doesn't change its value, and always gives the segment part of the full data item address).
在使用单个数据段(微型、小型和中型)的模型中，数据指针的长度为 16 位，只包含一个偏移量字段(DS 寄存器不改变其值，并始终提供完整数据项地址的段部分)。

• In models using more than one data segment (compact, large and huge), data pointers are 32 bits long, consisting of a 16−bit offset followed by a 16−bit segment. You should still be careful

not to modify DS in your routines without restoring it afterwards, but ES is free for you to use to access the contents of 32−bit data pointers you are passed.
在使用多个数据段(紧凑的、大的和巨大的)的模型中，数据指针的长度为 32 位，由 16 位偏移量和 16 位段组成。您仍然应该注意不要在例程中修改 DS 而不在事后还原它，但 ES 是免费的，您可以使用它来访问传递给您的 32 位数据指针的内容。

- The huge memory model allows single data items to exceed 64K in size. In all other memory models, you can access the whole of a data item just by doing arithmetic on the offset field of the pointer you are given, whether a segment field is present or not; in huge model, you have to be more careful of your pointer arithmetic.

巨大的内存模型允许单个数据项的大小超过 64k。在所有其他内存模型中，无论是否存在段字段字段，只要对给定的指针的偏移量字段进行算术运算，就可以访问整个数据项; 在大型模型中，必须更加注意指针算术。

- In most memory models, there is a *default* data segment, whose segment address is kept in DS throughout the program. This data segment is typically the same segment as the stack, kept in SS, so that functions' local variables (which are stored on the stack) and global data items can both be accessed easily without changing DS. Particularly large data items are typically stored in other segments. However, some memory models (though not the standard ones, usually) allow the assumption that SS and DS hold the same value to be removed. Be careful about functions' local variables in this latter case.

在大多数内存模型中，都有一个默认的数据段，其段地址在整个程序中都保留在 DS 中。这个数据段通常与堆栈相同，保存在 SS 中，因此函数的本地变量(存储在堆栈中)和全局数据项都可以很容易地访问，而无需更改 DS。特别大的数据项通常存储在其他段中。然而，一些内存模型(虽然通常不是标准模型)允许这样的假设，即 SS 和 DS 拥有相同的值被删除。在后一种情况下，要注意函数的局部变量。

In models with a single code segment, the segment is called _TEXT, so your code segment must also go by this name in order to be linked into the same place as the main code segment. In models with a single data segment, or with a default data segment, it is called _DATA.
在只有一个代码段的模型中，这个段称为 _text，因此您的代码段也必须使用这个名称，以便与主代码段链接到相同的位置。在只有一个数据段或者默认数据段的模型中，它被称为 _ DATA。

8.4.3 Function Definitions and Function Calls
8.4.3 函数定义和函数调用

The C calling convention in 16−bit programs is as follows. In the following description, the words *caller* and *callee* are used to denote the function doing the calling and the function which gets called.
16 位程序中的 c 调用约定如下。在下面的描述中，调用者和被调用者用来表示执行调用的函数和被调用的函数。

- The caller pushes the function's parameters on the stack, one after another, in reverse order (right to left, so that the first argument specified to the function is pushed last).

调用者将函数的参数一个接一个地按相反的顺序推送到堆栈上(从右到左，因此指定给函数的第一个参数最后被推送)。

- The caller then executes a `CALL` instruction to pass control to the callee. This `CALL` is either near or far depending on the memory model.

然后，调用方执行 CALL 指令将控制权传递给被调用方。这个 CALL 是近还是远取决于内存模型。

- The callee receives control, and typically (although this is not actually necessary, in functions which do not need to access their parameters) starts by saving the value of `SP` in `BP` so as to be able to use `BP` as a base pointer to find its parameters on the stack. However, the caller was probably doing this too, so part of the calling convention states that `BP` must be preserved by any C function. Hence the callee, if it is going to set up `BP` as a *frame pointer*, must push the previous value first.

被调用方接受控制，并且通常(尽管在不需要访问其参数的函数中实际上并不需要这样做)首先在 BP 中保存 SP 的值，以便能够使用 BP 作为基指针在堆栈中查找其参数。然而，调用者可能也在这么做，所以调用约定的一部分指出 BP 必须被任何 c 函数保留。因此，如果被调用者要将 BP 设置为一个帧指针，必须先推前一个值。

- The callee may then access its parameters relative to `BP`. The word at `[BP]` holds the previous value of `BP` as it was pushed; the next word, at `[BP+2]`, holds the offset part of the return address, pushed implicitly by `CALL`. In a small−model (near) function, the parameters start after that, at `[BP+4]`; in a large−model (far) function, the segment part of the return address lives at `[BP+4]`, and the parameters begin at `[BP+6]`. The leftmost parameter of the function, since it was pushed last, is accessible at this offset from `BP`; the others follow, at successively greater offsets. Thus, in a function such as `printf` which takes a variable number of parameters, the pushing of the parameters in reverse order means that the function knows where to find its first parameter, which tells it the number and type of the remaining ones.

然后被调用方可以访问其相对于 BP 的参数。单词 at [BP]保存前一个被推送的 BP 值; 下一个单词 at [BP + 2]保存返回地址的偏移量部分，由 CALL 隐式推送。在小模型(近)函数中，参数从[BP + 4]开始; 在大模型(远)函数中，返回地址的段部分从[BP + 4]开始，参数从[BP + 6]开始。该函数的最左边的参数，因为它是最后推出的，可以在 BP 的这个偏移量处访问; 其他参数随后在相继更大的偏移量处访问。因此，在像 printf 这样的函数中，参数数量可变，按相反的顺序推送参数意味着函数知道在哪里找到它的第一个参数，这个参数告诉它剩余参数的数量和类型。

- The callee may also wish to decrease `SP` further, so as to allocate space on the stack for local variables, which will then be accessible at negative offsets from `BP`.

被调用方也可能希望进一步减少 SP，以便在堆栈上为局部变量分配空间，这样就可以在 BP 的负偏移量处访问这些变量。

- The callee, if it wishes to return a value to the caller, should leave the value in `AL`, `AX` or `DX:AX` depending on the size of the value. Floating−point results are sometimes (depending on the compiler) returned in `ST0`.

如果被调用方希望向调用方返回一个值，则应根据该值的大小将该值保留为 AL、 AX 或 DX: AX。浮点结果有时(取决于编译器)会在 st0 中返回。

- Once the callee has finished processing, it restores `SP` from `BP` if it had allocated local stack space, then pops the previous value of `BP`, and returns via `RETN` or `RETF` depending on memory model.

一旦被调用方完成处理，如果已经分配了本地堆栈空间，它将从 BP 恢复 SP，然后弹出 BP 的前一个值，并根据内存模型通过 RETN 或 RETF 返回。

- When the caller regains control from the callee, the function parameters are still on the stack, so it typically adds an immediate constant to `SP` to remove them (instead of executing a number of slow

POP instructions). Thus, if a function is accidentally called with the wrong number of parameters due to a prototype mismatch, the stack will still be returned to a sensible state since the caller, which *knows* how many parameters it pushed, does the removing.
当调用方从被调用方重新获得控制权时，函数参数仍然在堆栈上，因此它通常会向 SP 添加一个即时常量来删除它们(而不是执行许多缓慢的 POP 指令)。因此，如果由于原型不匹配而意外使用错误数量的参数调用函数，堆栈仍然会返回到合理的状态，因为调用者知道自己推送了多少参数，所以会进行删除。

It is instructive to compare this calling convention with that for Pascal programs (described in section 8.5.1). Pascal has a simpler convention, since no functions have variable numbers of parameters. Therefore the callee knows how many parameters it should have been passed, and is able to deallocate them from the stack itself by passing an immediate argument to the RET or RETF instruction, so the caller does not have to do it. Also, the parameters are pushed in left−to−right order, not right−to−left, which means that a compiler can give better guarantees about sequence points without performance suffering.
比较这个调用约定和 Pascal 程序的调用约定(在第 8.5.1 节中描述)是有益的。Pascal 有一个更简单的约定，因为没有函数有可变数量的参数。因此，被调用方知道应该传递多少个参数，并且能够通过向 RET 或 RETF 指令传递一个即时参数从堆栈本身释放这些参数，因此调用方不必这样做。此外，参数按照从左到右的顺序推送，而不是从右到左，这意味着编译器可以更好地保证序列点，而不会影响性能。

Thus, you would define a function in C style in the following way. The following example is for small model:
因此，您可以通过以下方式定义 c 样式的函数:

```
global  _myfunc
Global _ myfunc

_myfunc:
我的功能:
        push    bp
        按压血压
        mov     bp,sp
        血压，血压
```

```
sub       sp,0x40           ; 64 bytes of local stack space
字幕      Sp，0 x40          64 字节的本地堆栈空间
mov       bx,[bp+4]         ; first parameter to function
动起来    Bx [ bp + 4]      ; 函数的第一个参数
; some more code
更多的代码
mov       sp,bp             ; undo "sub sp,0x40" above
动起来    Sp，bp            ; 撤销上面的“ sub sp，0 x40”
pop       bp
啪        血压
ret
后悔
```

For a large−model function, you would replace `RET` by `RETF`, and look for the first parameter at `[BP+6]` instead of `[BP+4]`. Of course, if one of the parameters is a pointer, then the offsets of *subsequent* parameters will change depending on the memory model as well: far pointers take up four bytes on the stack when passed as a parameter, whereas near pointers take up two.
对于大型模型函数，可以用 RETF 代替 RET，并在[BP + 6]而不是[BP + 4]中查找第一个参数。当然，如果其中一个参数是指针，那么后续参数的偏移量也会随着内存模型的不同而改变: 作为参数传递的远指针占用堆栈上的 4 个字节，而近指针占用 2 个字节。

At the other end of the process, to call a C function from your assembly code, you would do something like this:
在这个过程的另一端，从汇编代码调用 c 函数，你可以这样做:

```
extern  _printf
外部打印

        ; and then, further down...
        然后，再往下。

        push      word  [myint]         ; one of    my integer variables
        推        字    [ myint ]       ; 其中之一  我的整数变量
                        mystring
        push      word  Mystring 我的      pointer into     my data segment
        推        字    线              ;  指针指向         我的数据段
        call      _printf
        打电话    _ printf
        add       sp,byte 4                'byte'    saves   space
        添加      Sp，字节 4            ;  字节      保存    太空

        ; then those data items...
        那么这些数据项目。

segment _DATA
段 _ 数据

myint         dw    1234
Myint dw 1234
mystring      db    'This number -> %d <- should be 1234',10,0
Mystringdb'这个数字->% d <-应该是 1234'，10，0
```

This piece of code is the small−model assembly equivalent of the C code
这段代码是 c 代码的小型程序集等价物

```
    int myint = 1234;
    Int myint = 1234;
```

```
printf("This number -> %d <- should be 1234\n", myint);
Printf (“这个数字->% d <-应该是 1234 n”, myint) ;
```

In large model, the function−call code might look more like this. In this example, it is assumed that DS already holds the segment base of the segment _DATA. If not, you would have to initialize it first.
在大型模型中，函数调用代码可能看起来更像这样。在这个例子中，假设 DS 已经拥有段 _ DATA 的段基。如果没有，你必须首先初始化它。

```
push      word [myint]
推        单词[ myint ]
push      word seg mystring          Now push the segment, and...
推        单词 seg mystring        ; 现在按下这段，然后..。
push      word mystring              ... offset of "mystring"
推        字谜字符串               ; ... “ mystring”的偏移量
call      far _printf
打电话    打印
add      sp,byte 6
添加     Sp，字节 6
```

The integer value still takes up one word on the stack, since large model does not affect the size of the int data type. The first argument (pushed last) to printf, however, is a data pointer, and therefore has to contain a segment and offset part. The segment should be stored second in memory, and therefore must be pushed first. (Of course, PUSH DS would have been a shorter instruction than PUSH WORD SEG mystring, if DS was set up as the above example assumed.) Then the actual call becomes a far call, since functions expect far calls in large model; and SP has to be increased by 6 rather than 4 afterwards to make up for the extra word of parameters.
整数值在堆栈上仍然占用一个字，因为大型模型不影响 int 数据类型的大小。然而，printf 的第一个参数(最后一个)是一个数据指针，因此必须包含一个段和偏移量部分。段应该存储在内存中的第二位，因此必须首先推送。(当然，如果按照上面的例子设置 DS 的话，PUSH DS 会比 PUSH WORD SEG mystring 更短。)然后实际的调用就变成了 far 调用，因为函数期望在大型模型中进行 far 调用; 之后 SP 必须增加 6 而不是 4，以弥补额外的参数字。

8.4.4 Accessing Data Items
8.4.4 访问数据项目

To get at the contents of C variables, or to declare variables which C can access, you need only declare the names as `GLOBAL` or `EXTERN`. (Again, the names require leading underscores, as stated in section 8.4.1.) Thus, a C variable declared as `int i` can be accessed from assembler as
要获取 c 变量的内容，或者声明 c 可以访问的变量，你只需要将名称声明为 GLOBAL 或 EXTERN。(同样，名称需要前导下划线，如第 8.4.1 节所述因此，一个被声明为 int i 的 c 变量可以被汇编程序访问为

```
extern _i
```
译者注:

```
        mov ax,[_i]
        Mov ax, [ _ i ]
```

And to declare your own integer variable which C programs can access as `extern int j`, you do this (making sure you are assembling in the `_DATA` segment, if necessary):
并声明自己的整数变量，c 程序可以作为外部 int j 访问这个变量，您可以这样做(如果必要的话，确保在 _data 段中进行组装) :

```
global  _j
全局 _ j

_j      dw      0
_ j dw 0
```

To access a C array, you need to know the size of the components of the array. For example, `int` variables are two bytes long, so if a C program declares an array as `int a[10]`, you can access `a[3]` by coding `mov ax,[_a+6]`. (The byte offset 6 is obtained by multiplying the desired array index, 3, by the size of the array element, 2.) The sizes of the C base types in 16−bit compilers are: 1 for `char`, 2 for `short` and `int`, 4 for `long` and `float`, and 8 for `double`.
要访问一个 c 数组，你需要知道数组组件的大小。例如，int 变量有两个字节长，因此如果一个 c 程序声明一个数组为 int a [10]，您可以通过编码 mov ax [_ a + 6]来访问一个[3]。(字节偏移量 6 是通过将所需的数组索引 3 乘以数组元素的大小 2 得到的 16 位编译器中 c 基类型的大小是: 1 表示 char，2 表示 short 和 int，4 表示 long 和 float，8 表示 double。

To access a C data structure, you need to know the offset from the base of the structure to the field you are interested in. You can either do this by converting the C structure definition into a NASM structure definition (using `STRUC`), or by calculating the one offset and using just that.
要访问一个 c 数据结构，你需要知道从结构的底部到你感兴趣的字段的偏移量。您可以通过将 c 结构定义转换为 NASM 结构定义(使用 STRUC)来实现这一点，或者通过计算一个偏移量并仅使用它来实现这一点。

To do either of these, you should read your C compiler's manual to find out how it organizes data structures. NASM gives no special alignment to structure members in its own `STRUC` macro, so you have to specify alignment yourself if the C compiler generates it. Typically, you might find that a structure like
要做到这两点，您应该阅读 c 编译器手册，了解它是如何组织数据结构的。NASM 在它自己的 STRUC 宏中没有给出结构成员的特殊对齐方式，所以如果 c 编译器生成它，你必须自己指定对齐方式。通常情况下，你可能会发现像

```
struct {
结构{
    char c;
    字符 c;
    int i;
    Int i;
} foo;
翻译
```

might be four bytes long rather than three, since the `int` field would be aligned to a two−byte boundary. However, this sort of feature tends to be a configurable option in the C compiler, either using command−line options or `#pragma` lines, so you have to find out how your own compiler does it.
可能是四个字节而不是三个字节，因为 int 字段将对齐到两个字节的边界。然而，这种特性往往是 c 编译器中的一个可配置选项，要么使用命令行选项，要么使用 # pragma 行，因此您必须了解自己的编译器是如何做到这一点的。

8.4.5 `c16.mac`: Helper Macros for the 16−bit C Interface
8.4.5 c16.mac: 16 位 c 接口的辅助宏

Included in the NASM archives, in the `misc` directory, is a file `c16.mac` of macros. It defines three macros: `proc`, `arg` and `endproc`. These are intended to be used for C−style procedure definitions, and they automate a lot of the work involved in keeping track of the calling convention.
在 NASM 归档文件中的 misc 目录中包含一个宏文件 c16.mac。它定义了三个宏: proc，arg 和 endproc。它们用于 c 风格的过程定义，它们自动化了许多跟踪调用约定所涉及的工作。

(An alternative, TASM compatible form of `arg` is also now built into NASM's preprocessor. See section 4.8 for details.)
(一种替代的，TASM 兼容的 arg 形式现在也被内置到 NASM 的预处理器中。详见第 4.8 节

An example of an assembly function using the macro set is given here:
这里给出了一个使用宏集合的汇编函数示例:

```
proc    _nearproc
Proc _ nearproc

%$i        arg
% $i       Arg
%$j        arg
% $j       Arg
                     ax,[bp
           mov       Ax，      + %$i]
           动起来    [ bp      +% $i ]
                     bx,[bp
           mov       Bx，      + %$j]
           动起来    [ bp      +% $j ]
           add       ax,[bx]
           添加      Ax，［ bx ]
```

```
endproc
Endproc
```

This defines `_nearproc` to be a procedure taking two arguments, the first (`i`) an integer and the second (`j`) a pointer to an integer. It returns `i + *j`.
这将 _nearproc 定义为带有两个参数的过程，第一个(i)是整数，第二个(j)是指向整数的指针。它返回 i + * j。

Note that the `arg` macro has an `EQU` as the first line of its expansion, and since the label before the macro call gets prepended to the first line of the expanded macro, the `EQU` works, defining `%$i` to be an offset from `BP`. A context−local variable is used, local to the context pushed by the `proc` macro and popped by the `endproc` macro, so that the same argument name can be used in later procedures. Of course, you don't *have* to do that.
注意，arg 宏有一个 EQU 作为其展开的第一行，并且由于宏调用之前的标签被放在展开的宏的第一行之前，因此 EQU 起作用，将% $i 定义为 BP 的偏移量。使用一个 context-local 变量，该变量局部于 proc 宏推送的上下文，并由 endproc 宏弹出，以便在以后的过程中使用相同的参数名。当然，你不必这样做。

The macro set produces code for near functions (tiny, small and compact−model code) by default. You can have it generate far functions (medium, large and huge−model code) by means of coding `%define FARCODE`. This changes the kind of return instruction generated by `endproc`, and also changes the starting point for the argument offsets. The macro set contains no intrinsic dependency on whether data pointers are far or not.
宏集默认情况下生成近似函数的代码(微型、小型和紧凑型代码)。你可以让它生成远函数(中型、大型和巨型模型代码)，方法是编码% define FARCODE。这改变了 endproc 生成的返回指令的类型，也改变了参数偏移的起点。宏集不包含数据指针是否远的内在依赖关系。

`arg` can take an optional parameter, giving the size of the argument. If no size is given, 2 is assumed, since it is likely that many function parameters will be of type `int`.
`Arg` 可以接受一个可选参数，给出参数的大小。如果没有给出大小，则假定为 2，因为很多函数参数可能是 `int` 类型的。

The large−model equivalent of the above function would look like this:
上述函数的大型模型等价物是这样的:

```
%define FARCODE
% 定义 FARCODE

proc     _farproc
译注:    Farproc
%$i      arg
% $i     Arg
%$j      arg
% $j     Arg      4
         mov      ax,[bp + %$i]
         动起来   Ax，[ bp +% $i ]
         mov      bx,[bp + %$j]
         动起来   Bx，[ bp +% $j ]
                  es,[bp + %$j + 2]
         mov      英国石油公司股价上
         动起来   涨
         add      ax,[bx]
         添加     Ax，[ bx ]

endproc
Endproc
```

This makes use of the argument to the `arg` macro to define a parameter of size 4, because `j` is now a far pointer. When we load from `j`, we must load a segment and an offset.
这就利用了 arg 宏的参数来定义一个大小为 4 的参数，因为 j 现在是一个 far 指针。当我们从 j 加载时，我们必须加载一个段和一个偏移量。

8.5 Interfacing to Borland Pascal Programs
8.5 与 Borland Pascal 程序的接口

Interfacing to Borland Pascal programs is similar in concept to interfacing to 16−bit C programs. The differences are:
Borland Pascal 程序的接口在概念上类似于 16 位 c 程序的接口。区别在于:

• The leading underscore required for interfacing to C programs is not required for Pascal.
与 c 程序接口所需的前导下划线对于 Pascal 来说不是必需的。

• The memory model is always large: functions are far, data pointers are far, and no data item can be more than 64K long. (Actually, some functions are near, but only those functions that are local to a Pascal unit and never called from outside it. All assembly functions that Pascal calls, and all Pascal functions that assembly routines are able to call, are far.) However, all static data declared in a Pascal program goes into the default data segment, which is the one whose segment address will be in `DS` when control is passed to your assembly code. The only things that do not live in the default data segment are local variables (they live in the stack segment) and dynamically allocated variables. All data *pointers*, however, are far.
内存模型总是很大: 函数很远，数据指针很远，任何数据项的长度都不能超过 64k。(实际上，有些函数在附近，但是只有那些在 Pascal 单元内部并且从来没有从外部调用过的函数。所有 Pascal 调用的汇编函数，以及所有汇编例程能够调用的 Pascal 函数都很远然而，在 Pascal 程序中声明的所有静态数据都将进入默认数据段，当控制权传递给汇编代码时，默认数据段的段地址将在 DS 中。唯一不在默认数据段中的是本地变量(它们在堆栈段中)和动态分配的变量。然而，所有的数据指针都很远。

• The function calling convention is different – described below.
函数调用约定是不同的，如下所述。

• Some data types, such as strings, are stored differently.
一些数据类型，比如字符串，存储方式不同。

- There are restrictions on the segment names you are allowed to use – Borland Pascal will ignore code or data declared in a segment it doesn't like the name of. The restrictions are described below.
对于允许使用的段名有一些限制—— Borland Pascal 会忽略在它不喜欢的段名中声明的代码或数据。这些限制如下所述。

8.5.1 The Pascal Calling Convention
8.5.1 帕斯卡呼叫约定

The 16−bit Pascal calling convention is as follows. In the following description, the words *caller* and *callee* are used to denote the function doing the calling and the function which gets called.
16 位 Pascal 调用约定如下。在下面的描述中，调用者和被调用者用来表示执行调用的函数和被调用的函数。

- The caller pushes the function's parameters on the stack, one after another, in normal order (left to right, so that the first argument specified to the function is pushed first).
调用者按照正常顺序(从左到右，这样首先推送指定给函数的第一个参数)一个接一个地推送堆栈上的函数参数。

- The caller then executes a far `CALL` instruction to pass control to the callee.
然后调用者执行一个远程 CALL 指令，将控制权传递给被调用者。

- The callee receives control, and typically (although this is not actually necessary, in functions which do not need to access their parameters) starts by saving the value of `SP` in `BP` so as to be able to use `BP` as a base pointer to find its parameters on the stack. However, the caller was probably doing this too, so part of the calling convention states that `BP` must be preserved by any function. Hence the callee, if it is going to set up `BP` as a frame pointer, must push the previous value first.
被调用方接受控制，并且通常(尽管在不需要访问其参数的函数中实际上并不需要这样做)首先在 BP 中保存 SP 的值，以便能够使用 BP 作为基指针在堆栈中查找其参数。然而，调用者可能也在这么做，所以调用约定的一部分指出 BP 必须被任何函数保留。因此，如果被调用者要将 BP 设置为一个帧指针，必须首先推送前一个值。

- The callee may then access its parameters relative to `BP`. The word at `[BP]` holds the previous value of `BP` as it was pushed. The next word, at `[BP+2]`, holds the offset part of the return address, and the next one at `[BP+4]` the segment part. The parameters begin at `[BP+6]`. The rightmost parameter of the function, since it was pushed last, is accessible at this offset from `BP`; the others follow, at successively greater offsets.
然后被调用方可以访问其相对于 BP 的参数。在[BP]中的单词保持 BP 之前被推送时的值。下一个单词 at [BP + 2]包含返回地址的偏移量部分，下一个单词 at [BP + 4]包含段部分。参数从[BP + 6]开始。该函数的最右边的参数，因为它是最后推出的，可以在 BP 的这个偏移量处访问; 其他参数随后在相继更大的偏移量处访问。

- The callee may also wish to decrease `SP` further, so as to allocate space on the stack for local variables, which will then be accessible at negative offsets from `BP`.
被调用方也可能希望进一步减少 SP，以便在堆栈上为局部变量分配空间，这样就可以在 BP 的负偏移量处访问这些变量。

- The callee, if it wishes to return a value to the caller, should leave the value in `AL`, `AX` or `DX:AX` depending on the size of the value. Floating−point results are returned in `ST0`. Results of type `Real` (Borland's own custom floating−point data type, not handled directly by the FPU) are returned in `DX:BX:AX`. To return a result of type `String`, the caller pushes a pointer to a temporary string before pushing the parameters, and the callee places the returned string value at that location. The pointer is not a parameter, and should not be removed from the stack by the `RETF` instruction.
如果被调用方希望向调用方返回一个值，则应根据该值的大小将该值保留为 AL、 AX 或 DX: AX。浮点结果在 st0 中返回。Real 类型的结果(Borland 自己的自定义浮点数据类型，不由 FPU 直接处理)在 DX: BX: AX 中返回。为了返回 String 类型的结果，调用方在推送参数之前将指针推送到临时字符串，并且被调用方将返回的字符串值放在该位置。指针不是一个参数，不应该被 RETF 指令从堆栈中移除。

- Once the callee has finished processing, it restores `SP` from `BP` if it had allocated local stack space, then pops the previous value of `BP`, and returns via `RETF`. It uses the form of `RETF` with an

immediate parameter, giving the number of bytes taken up by the parameters on the stack. This causes the parameters to be removed from the stack as a side effect of the return instruction.
一旦被调用方完成处理，如果已经分配了本地堆栈空间，它将从 BP 恢复 SP，然后弹出 BP 的前一个值，并通过 RETF 返回。它使用 RETF 的形式和一个即时参数，给出堆栈参数占用的字节数。作为返回指令的一个副作用，这会导致参数从堆栈中被移除。

• When the caller regains control from the callee, the function parameters have already been removed from the stack, so it needs to do nothing further.
当调用者从被调用者那里重新获得控制权时，函数参数已经从堆栈中被移除了，所以它不需要再做任何事情。

Thus, you would define a function in Pascal style, taking two `Integer`–type parameters, in the following way:
因此，您可以通过以下方式定义 Pascal 样式的函数，接受两个 Integer 类型的参数:

```
global  myfunc
全局 myfunc

myfunc: push    bp
用力            血压
        mov     bp,sp
        动起来  Bp，sp
        sub     sp,0x40         ; 64 bytes of local stack space
        字幕    Sp，0 x40       64 字节的本地堆栈空间
        mov     bx,[bp+8]       ; first parameter to function
        动起来  Bx [ bp + 8]    ; 函数的第一个参数
        mov     bx,[bp+6]       ; second parameter to function
        动起来  Bx [ bp + 6]    函数的第二个参数
        ; some more code
        更多的代码
        mov     sp,bp           ; undo "sub sp,0x40" above
        动起来  Sp，bp          ; 撤销上面的“ sub sp，0 x40”
        pop     bp
        啪      血压
        retf                    ; total size of params is 4
        翻译    4               ; 参数的总大小是 4
```

At the other end of the process, to call a Pascal function from your assembly code, you would do something like this:
在流程的另一端，从汇编代码调用 Pascal 函数，您可以这样做:

```
extern  SomeFunc
Extern SomeFunc

        ; and then, further down...
        然后，再往下。

        push    word seg mystring       ; Now   push the segment, and...
        推      单词 seg mystring       现在    推动这个环节，然后..。
        push    word mystring             ...   offset of "mystring"
        推      字谜字符串              ; ...   “ mystring”的偏移量
        push    word [myint]              one   of my variables
        推      单词[ myint ]           ; 一    我的变量
        call    far SomeFunc
        打电话  Far SomeFunc
```

This is equivalent to the Pascal code
这相当于 Pascal 代码

```
procedure SomeFunc(String: PChar; Int: Integer);
    SomeFunc(@mystring, myint);
过程 SomeFunc (String: PChar; Int: Integer) ;
```

8.5.2 Borland Pascal Segment Name Restrictions
8.5.2 Borland Pascal 段名限制

Since Borland Pascal’s internal unit file format is completely different from OBJ, it only makes a very sketchy job of actually reading and understanding the various information contained in a real OBJ file when it links that in. Therefore an object file intended to be linked to a Pascal program must obey a number of restrictions:
由于 Borland Pascal 的内部单元文件格式与 OBJ 是完全不同的，所以当 OBJ 文件链接到 OBJ 文件时，实际上读取和理解 OBJ 文件中包含的各种信息只是一个非常粗略的工作。因此，一个想要链接到 Pascal 程序的目标文件必须遵守一些限制:

• Procedures and functions must be in a segment whose name is either CODE, CSEG, or something ending in _TEXT.
过程和函数必须在一个段中，这个段的名字要么是 CODE，CSEG，要么是以 _ TEXT 结尾的东西。

• initialized data must be in a segment whose name is either CONST or something ending in _DATA.
初始化的数据必须位于一个段中，该段的名称要么是 CONST，要么是以 _ DATA 结尾的内容。

• Uninitialized data must be in a segment whose name is either DATA, DSEG, or something ending in
未初始化的数据必须位于一个段中，该段的名称要么是 DATA，DSEG，要么是以
_BSS.
生物安全系统。

• Any other segments in the object file are completely ignored. GROUP directives and segment attributes are also ignored.
目标文件中的任何其他段都被完全忽略。 GROUP 指令和段属性也被忽略。

8.5.3 Using c16.mac With Pascal Programs
8.5.3 使用 c16.mac 和 Pascal 程序

The c16.mac macro package, described in section 8.4.5, can also be used to simplify writing functions to be called from Pascal programs, if you code %define PASCAL. This definition ensures that functions are far (it implies FARCODE), and also causes procedure return instructions to be generated with an operand.

如果您的代码% 定义 PASCAL，那么第 8.4.5 节中描述的 c16.mac 宏包也可以用来简化从 PASCAL 程序中调用的编写函数。这个定义确保函数是远的(这意味着 FARCODE)，也使得过程返回指令用操作数生成。

Defining `PASCAL` does not change the code which calculates the argument offsets; you must declare your function’s arguments in reverse order. For example:
定义 PASCAL 不会更改计算参数偏移量的代码; 必须按相反的顺序声明函数的参数。例如:

```
%define PASCAL
% 定义 PASCAL

proc    _pascalproc
译注:   帕斯卡普罗克
%$j     arg 4
% $j    Arg 4
%$i     arg
% $i    Arg
        mov     ax,[bp + %$i]
        动起来  Ax，[ bp +% $i ]
        mov     bx,[bp + %$j]
        动起来  Bx，[ bp +% $j ]
                es,[bp + %$j + 2]
        mov     英国石油公司股价上
        动起来  涨
        add     ax,[bx]
        添加    Ax，[ bx ]

endproc
Endproc
```

This defines the same routine, conceptually, as the example in section 8.4.5: it defines a function taking two arguments, an integer and a pointer to an integer, which returns the sum of the integer and the
这在概念上定义了与第 8.4.5 节中的示例相同的例程: 它定义了一个函数，该函数带有两个参数，一个整数和一个指向整数的指针，该函数返回整数和

contents of the pointer. The only difference between this code and the large−model C version is that `PASCAL` is defined instead of `FARCODE`, and that the arguments are declared in reverse order.
指针的内容。此代码与大型模型 c 版本之间的唯一区别在于，定义的是 PASCAL 而不是 FARCODE，并且参数的声明顺序是相反的。

Chapter 9: Writing 32−bit Code (Unix, Win32, DJGPP)
第九章: 编写 32 位代码(Unix，Win32，DJGPP)

This chapter attempts to cover some of the common issues involved when writing 32−bit code, to run under Win32 or Unix, or to be linked with C code generated by a Unix−style C compiler such as DJGPP. It covers how to write assembly code to interface with 32−bit C routines, and how to write position−independent code for shared libraries.
本章试图涵盖编写 32 位代码、在 win32 或 Unix 下运行、或与 Unix 风格的 c 编译器(如 DJGPP)生成的 c 代码链接时所涉及的一些常见问题。它涵盖了如何编写汇编代码来接口 32 位 c 例程，以及如何编写位置无关的共享库代码。

Almost all 32−bit code, and in particular all code running under `Win32`, `DJGPP` or any of the PC Unix variants, runs in *flat* memory model. This means that the segment registers and paging have already been set up to give you the same 32−bit 4Gb address space no matter what segment you work relative to, and that you should ignore all segment registers completely. When writing flat−model application code, you never need to use a segment override or modify any segment register, and the code−section addresses you pass to `CALL` and `JMP` live in the same address space as the data−section addresses you access your variables by and the stack−section addresses you access local variables and procedure parameters by. Every address is 32 bits long and contains only an offset part.
几乎所有 32 位代码，特别是在 Win32、 DJGPP 或任何 PC Unix 变体下运行的所有代码，都运行在平面内存模型中。这意味着段寄存器和分页已经被设置为无论您相对于哪个段工作，都提供相同的 32 位 4gb 地址空间，并且您应该完全忽略所有段寄存器。在编写平面模型应用程序代码时，您永远不需要使用段覆盖或修改任何段寄存器，您传递给 CALL 和 JMP 的代码段地址与您访问变量的数据段地址和您访问本地变量和过程参数的堆栈段地址位于同一地址空间中。每个地址长 32 位，只包含一个偏移量部分。

9.1 Interfacing to 32−bit C Programs
9.1 与 32 位 c 程序的接口

A lot of the discussion in section 8.4, about interfacing to 16−bit C programs, still applies when working in 32 bits. The absence of memory models or segmentation worries simplifies things a lot.
8.4 节中关于 16 位 c 程序接口的许多讨论，在 32 位工作时仍然适用。内存模型的缺失或者对分割的担忧使事情简化了很多。

9.1.1 External Symbol Names
9.1.1 外部符号名称

Most 32−bit C compilers share the convention used by 16−bit compilers, that the names of all global symbols (functions or data) they define are formed by prefixing an underscore to the name as it appears in the C program. However, not all of them do: the `ELF` specification states that C symbols do *not* have a leading underscore on their assembly−language names.
大多数 32 位 c 编译器都遵循 16 位编译器使用的惯例，即它们定义的所有全局符号(函数或数据)的名称都是通过在名称前面加上一个下划线来形成的，就像它在 c 程序中出现的那样。然而，并不是所有的符号都这样做: ELF 规范声明 c 符号在汇编语言名称上没有前置下划线。

The older Linux `a.out` C compiler, all `Win32` compilers, `DJGPP`, and `NetBSD` and `FreeBSD`, all use the leading underscore; for these compilers, the macros `cextern` and `cglobal`, as given in section 8.4.1, will still work. For `ELF`, though, the leading underscore should not be used.
旧的 Linux a.out c 编译器，所有 win32 编译器，DJGPP，NetBSD 和 FreeBSD，都使用前面的下划线; 对于这些编译器，宏 cextern 和 cglobal，如第 8.4.1 节所示，仍然可以工作。然而，对于 ELF 来说，前面的下划线不应该被使用。

See also section 2.1.28.
参见第 2.1.28 节。

9.1.2 Function Definitions and Function Calls

9.1.2 函数定义和函数调用

The C calling convention in 32-bit programs is as follows. In the following description, the words *caller* and *callee* are used to denote the function doing the calling and the function which gets called.
32 位程序中的 c 调用约定如下。在下面的描述中，调用者和被调用者用来表示执行调用的函数和被调用的函数。

- The caller pushes the function's parameters on the stack, one after another, in reverse order (right to left, so that the first argument specified to the function is pushed last).
调用者将函数的参数一个接一个地按相反的顺序推送到堆栈上(从右到左，因此指定给函数的第一个参数最后被推送)。

- The caller then executes a near `CALL` instruction to pass control to the callee.
然后调用者执行一个近似的 CALL 指令，将控制权传递给被调用者。

- The callee receives control, and typically (although this is not actually necessary, in functions which do not need to access their parameters) starts by saving the value of `ESP` in `EBP` so as to be able to use `EBP` as a base pointer to find its parameters on the stack. However, the caller was probably doing this too, so part of the calling convention states that `EBP` must be preserved by any C function. Hence the callee, if it is going to set up `EBP` as a frame pointer, must push the previous value first.
被调用方接受控制，并且通常(尽管在不需要访问其参数的函数中实际上并不需要这样做)首先在 EBP 中保存 ESP 的值，以便能够使用 EBP 作为基本指针在堆栈中查找其参数。然而，调用者可能也在这么做，所以调用约定的一部分指出 EBP 必须被任何 c 函数保留。因此，如果被调用者要将 EBP 设置为一个帧指针，必须首先推送前一个值。

- The callee may then access its parameters relative to `EBP`. The doubleword at `[EBP]` holds the previous value of `EBP` as it was pushed; the next doubleword, at `[EBP+4]`, holds the return address, pushed implicitly by `CALL`. The parameters start after that, at `[EBP+8]`. The leftmost parameter of the function, since it was pushed last, is accessible at this offset from `EBP`; the others follow, at successively greater offsets. Thus, in a function such as `printf` which takes a variable number of parameters, the pushing of the parameters in reverse order means that the function knows where to find its first parameter, which tells it the number and type of the remaining ones.
然后被调用方可以访问相对于 EBP 的参数。在[EBP]处的双字保存了 EBP 的前一个值，而在[EBP + 4]处的下一个双字保存了 CALL 隐式推送的返回地址。参数在那之后开始，在[EBP + 8]。函数的最左边的参数，因为它是最后推出的，可以在 EBP 的这个偏移量处访问; 其他参数随后，在相继更大的偏移量处。因此，在像 printf 这样的函数中，参数数量可变，按相反的顺序推送参数意味着函数知道在哪里找到它的第一个参数，这个参数告诉它剩余参数的数量和类型。

- The callee may also wish to decrease `ESP` further, so as to allocate space on the stack for local variables, which will then be accessible at negative offsets from `EBP`.

被调用者也可能希望进一步减少 ESP，以便在堆栈上为局部变量分配空间，然后可以从 EBP 的负偏移量访问这些空间。

- The callee, if it wishes to return a value to the caller, should leave the value in `AL`, `AX` or `EAX` depending on the size of the value. Floating−point results are typically returned in `ST0`.

如果被调用方希望向调用方返回一个值，则应根据该值的大小将该值保留在 AL、 AX 或 EAX 中。浮点结果通常在 st0 中返回。

- Once the callee has finished processing, it restores `ESP` from `EBP` if it had allocated local stack space, then pops the previous value of `EBP`, and returns via `RET` (equivalently, `RETN`).

一旦被调用方完成了处理，如果它已经分配了本地堆栈空间，那么它将从 EBP 中恢复 ESP，然后弹出 EBP 的前一个值，并通过 RET (相当于 RETN)返回。

- When the caller regains control from the callee, the function parameters are still on the stack, so it typically adds an immediate constant to `ESP` to remove them (instead of executing a number of slow `POP` instructions). Thus, if a function is accidentally called with the wrong number of parameters due to a prototype mismatch, the stack will still be returned to a sensible state since the caller, which *knows* how many parameters it pushed, does the removing.

当调用者从被调用者那里重新获得控制权时，函数参数仍然在堆栈上，因此它通常会向 ESP 添加一个即时常量来删除它们(而不是执行许多缓慢的 POP 指令)。因此，如果由于原型不匹配而意外使用错误数量的参数调用函数，堆栈仍然会返回到合理的状态，因为调用者知道自己推送了多少参数，所以会进行删除。

There is an alternative calling convention used by Win32 programs for Windows API calls, and also for functions called *by* the Windows API such as window procedures: they follow what Microsoft calls the `__stdcall` convention. This is slightly closer to the Pascal convention, in that the callee clears the stack by passing a parameter to the `RET` instruction. However, the parameters are still pushed in right−to−left order.
Win32 程序对 Windows API 调用以及 Windows API 调用的函数(如窗口过程)使用了另一种调用约定: 它们遵循 Microsoft 所称的 _ _ stdcall 约定。这稍微接近于 Pascal 约定，因为调用方通过向 RET 指令传递一个参数来清除堆栈。然而，参数仍然按照从右到左的顺序被推送。

Thus, you would define a function in C style in the following way:
因此，您可以通过以下方式定义 c 样式的函数:

```
global  _myfunc
Global _ myfunc

_myfunc:
我的功能:
        push    ebp
        推      Ebp
        mov     ebp,esp
        动起来  Ebp，尤指
        sub     esp,0x40        ; 64 bytes of local stack space
        字幕    尤其是，0 x40   64 字节的本地堆栈空间
        mov     ebx,[ebp+8]     ; first parameter to function
        动起来  Ebx，[ ebp + 8]  ; 函数的第一个参数
        ; some more code
        更多的代码
        leave                   ; mov esp,ebp / pop ebp
        走吧                    Mov esp，ebp
        ret
        后悔
```

At the other end of the process, to call a C function from your assembly code, you would do something like this:

在这个过程的另一端，从汇编代码调用 c 函数，你可以这样做:

```
extern  _printf
外部打印

        ; and then, further down...
        然后，再往下。

        push    dword  [myint]      ; one of     my integer variables
        推      Dword  [ myint ]    ; 其中之一   我的整数变量
                       mystring
        push    dword  Mystring 我     pointer into    my data segment
        推      Dword  的线         ;  指针指向        我的数据段
        call    _printf
        打电话  _ printf
        add     esp,byte 8             'byte'   saves   space
        添加    尤其是字节 8        ;  字节     保存    太空

        ; then those data items...
        那么这些数据项目。

segment _DATA
段 _ 数据

myint       dd   1234
Myint dd 1234
mystring    db   'This number -> %d <- should be 1234',10,0
Mystringdb'这个数字->% d <-应该是 1234', 10,0
```

This piece of code is the assembly equivalent of the C code
这段代码相当于 c 代码的汇编

```
    int myint = 1234;
    Int myint = 1234;
    printf("This number -> %d <- should be 1234\n", myint);
    Printf (“这个数字->% d <-应该是 1234 n”, myint) ;
```

9.1.3 Accessing Data Items
9.1.3 访问数据项

To get at the contents of C variables, or to declare variables which C can access, you need only declare the names as `GLOBAL` or `EXTERN`. (Again, the names require leading underscores, as stated in section 9.1.1.) Thus, a C variable declared as `int i` can be accessed from assembler as
要获取 c 变量的内容，或者声明 c 可以访问的变量，只需将名称声明为 GLOBAL 或 EXTERN 即可。(同样，名称需要前导下划线，如第 9.1.1 节所述因此，一个被声明为 int i 的 c 变量可以被汇编程序访问为

```
extern _i
译者注:
mov eax,[_i]
动起来，[ _ i ]
```

And to declare your own integer variable which C programs can access as `extern int j`, you do this (making sure you are assembling in the `_DATA` segment, if necessary):
并声明自己的整数变量，c 程序可以作为外部 int j 访问这个变量，您可以这样做(如果必要的话，确保在 _data 段中进行组装) :

```
          global _j
          Global _ j
_j        dd 0
_ j dd 0
```

To access a C array, you need to know the size of the components of the array. For example, `int` variables are four bytes long, so if a C program declares an array as `int a[10]`, you can access `a[3]` by coding `mov ax,[_a+12]`. (The byte offset 12 is obtained by multiplying the desired array index, 3, by the size of the array element, 4.) The sizes of the C base types in 32−bit compilers are: 1 for `char`, 2 for `short`, 4 for `int`, `long` and `float`, and 8 for `double`. Pointers, being 32−bit addresses, are also 4 bytes long.
要访问 c 数组，您需要知道数组组件的大小。例如，int 变量有 4 个字节长，因此如果一个 c 程序声明一个数组为 int a [10] ，您可以通过编码 mov ax [_ a + 12]来访问一个[3]。(字节偏移量 12 是通过将所需的数组索引 3 乘以数组元素的大小 4 得到的 32 位编译器中 c 基类型的大小是: 1 表示 char，2 表示 short，4 表示 int，long 和 float，8 表示 double。指针是 32 位地址，也是 4 字节长。

To access a C data structure, you need to know the offset from the base of the structure to the field you are interested in. You can either do this by converting the C structure definition into a NASM structure definition (using `STRUC`), or by calculating the one offset and using just that.
要访问一个 c 数据结构，你需要知道从结构的底部到你感兴趣的字段的偏移量。您可以通过将 c 结构定义转换为 NASM 结构定义(使用 STRUC)来实现这一点，或者通过计算一个偏移量并仅使用它来实现这一点。

To do either of these, you should read your C compiler's manual to find out how it organizes data structures. NASM gives no special alignment to structure members in its own `STRUC` macro, so you have to specify alignment yourself if the C compiler generates it. Typically, you might find that a structure like
要做到这两点，您应该阅读 c 编译器手册，了解它是如何组织数据结构的。NASM 在它自己的 STRUC 宏中没有给出结构成员的特殊对齐方式，所以如果 c 编译器生成它，你必须自己指定对齐方式。通常情况下，你可能会发现像

```
struct {
结构{
    char c;
    字符 c;
    int i;
    Int i;
} foo;
翻译
```

might be eight bytes long rather than five, since the `int` field would be aligned to a four−byte boundary. However, this sort of feature is sometimes a configurable option in the C compiler, either using command−line options or `#pragma` lines, so you have to find out how your own compiler does it.
可能是 8 字节而不是 5 字节，因为整型字段将被对齐到一个 4 字节的边界。然而，这种特性有时是 c 编译器中的一个可配置选项，可以使用命令行选项或 # pragma 行，因此您必须了解自己的编译器是如何做到这一点的。

9.1.4 `c32.mac`: Helper Macros for the 32−bit C Interface
9.1.4 c32.mac: 32 位 c 接口的 Helper 宏

Included in the NASM archives, in the `misc` directory, is a file `c32.mac` of macros. It defines three macros: `proc`, `arg` and `endproc`. These are intended to be used for C−style procedure definitions, and they automate a lot of the work involved in keeping track of the calling convention.
在 NASM 归档文件中，在 misc 目录中，有一个宏文件 c32.mac。它定义了三个宏: proc，arg 和 endproc。它们用于 c 风格的过程定义，它们自动化了许多跟踪调用约定所涉及的工作。

An example of an assembly function using the macro set is given here:
这里给出了一个使用宏集合的汇编函数示例:

```
proc    _proc32
Proc _ proc32

%$i     arg
% $i    Arg
%$j     arg
% $j    Arg
        mov     eax,[ebp    + %$i]
        动起来  Eax，[ ebp  +% $i ]
        mov     ebx,[ebp    + %$j]
        动起来  电子病历    +% $j ]
        add     eax,[ebx]
        添加    Eax，[ ebx ]

endproc
Endproc
```

This defines `_proc32` to be a procedure taking two arguments, the first (`i`) an integer and the second (`j`) a pointer to an integer. It returns `i + *j`.
这将 _ proc32 定义为带有两个参数的过程，第一个(i)是整数，第二个(j)是指向整数的指针。它返回 i + * j。

Note that the `arg` macro has an `EQU` as the first line of its expansion, and since the label before the macro call gets prepended to the first line of the expanded macro, the `EQU` works, defining `%$i` to be an offset from `BP`. A context−local variable is used, local to the context pushed by the `proc` macro and popped by the `endproc` macro, so that the same argument name can be used in later procedures. Of course, you don't *have* to do that.
注意，arg 宏有一个 EQU 作为其展开的第一行，并且由于宏调用之前的标签被放在展开的宏的第一行之前，因此 EQU 起作用，将% $i 定义为 BP 的偏移量。使用一个 context-local 变量，该变量局部于 proc 宏推送的上下文，并由 endproc 宏弹出，以便在以后的过程中使用相同的参数名。当然，你不必这样做。

`arg` can take an optional parameter, giving the size of the argument. If no size is given, 4 is assumed, since it is likely that many function parameters will be of type `int` or pointers.
Arg 可以接受一个可选参数，给出参数的大小。如果没有给出大小，则假定为 4，因为很多函数参数可能是 int 或指针类型的。

9.2 Writing NetBSD/FreeBSD/OpenBSD and Linux/ELF Shared Libraries
9.2 编写 NetBSD/FreeBSD/OpenBSD 和 Linux/ELF 共享库

`ELF` replaced the older `a.out` object file format under Linux because it contains support for position−independent code (PIC), which makes writing shared libraries much easier. NASM supports the `ELF` position−independent code features, so you can write Linux `ELF` shared libraries in NASM.
ELF 取代了 Linux 下旧的 a.out 对象文件格式，因为它支持位置无关代码(PIC) ，这使得编写共享库变得更加容易。NASM 支持 ELF 位置无关代码特性，所以你可以在 NASM 中编写 Linux ELF 共享库。

NetBSD, and its close cousins FreeBSD and OpenBSD, take a different approach by hacking PIC support into the `a.out` format. NASM supports this as the `aoutb` output format, so you can write BSD shared libraries in NASM too.
NetBSD，以及它的近亲 FreeBSD 和 OpenBSD，采取了不同的方法，将 PIC 支持黑入 a.out 格式。NASM 支持 aoutb 输出格式，所以你也可以用 NASM 编写 BSD 共享库。

The operating system loads a PIC shared library by memory−mapping the library file at an arbitrarily chosen point in the address space of the running process. The contents of the library's code section must therefore not depend on where it is loaded in memory.
操作系统通过内存映射库文件到运行进程地址空间中任意选择的点来加载 PIC 共享库。因此，库代码段的内容必须不依赖于它在内存中的加载位置。

Therefore, you cannot get at your variables by writing code like this:
因此，你不能通过写这样的代码来获取你的变量:

```
        mov     eax,[myvar]             ; WRONG
        错
```

Instead, the linker provides an area of memory called the *global offset table*, or GOT; the GOT is situated at a constant distance from your library's code, so if you can find out where your library is loaded (which is typically done using a `CALL` and `POP` combination), you can obtain the address of the GOT, and you can then load the addresses of your variables out of linker−generated entries in the GOT.
相反，链接器提供了一个称为全局偏移表(global offset table，简称 GOT)的内存区域; GOT 位于与库代码的恒定距离处，因此如果您能找到库的加载位置(通常使用 CALL 和 POP 组合完成) ，则可以获得 GOT 的地址，然后可以从 GOT 中链接器生成的条目中加载变量的地址。

The *data* section of a PIC shared library does not have these restrictions: since the data section is writable, it has to be copied into memory anyway rather than just paged in from the library file, so as long as it's being copied it can be relocated too. So you can put ordinary types of relocation in the data section without too much worry (but see section 9.2.4 for a caveat).
PIC 共享库的数据部分没有这些限制: 由于数据部分是可写的，因此无论如何都必须将其复制到内存中，而不是仅仅从库文件中分页，所以只要它被复制，它也可以被重新定位。因此，你可以把普通类型的重定位放在数据部分，而不用太担心(但请参阅第 9.2.4 节的警告)。

9.2.1 Obtaining the Address of the GOT
9.2.1 获取 GOT 的地址

Each code module in your shared library should define the GOT as an external symbol:
共享库中的每个代码模块都应该将 GOT 定义为一个外部符号:

```
extern _GLOBAL_OFFSET_TABLE_ ; in ELF extern
__GLOBAL_OFFSET_TABLE_ ; in BSD a.out
在 ELF extern _ GLOBAL _ offset _ table _; 在
BSD a.out
```

At the beginning of any function in your shared library which plans to access your data or BSS sections, you must first calculate the address of the GOT. This is typically done by writing the function in this form:
在共享库中任何计划访问数据或 BSS 部分的函数开始时，必须首先计算 GOT 的地址。这通常是通过以下形式编写函数来完成的:

```
func:   push    ebp
函数:   推      Ebp
        mov     ebp,esp
        动起来  Ebp，尤指
        push    ebx
        推      Ebx
        call    .get_GOT
        打电话  打电话
.get_GOT:
得到 -- 得到:
        pop     ebx
        啪      Ebx
        add     ebx,_GLOBAL_OFFSET_TABLE_+$$−.get_GOT wrt ..gotpc
        添加    Ebx，_ global _ offset _ table _ + $- . get _ got wrt. . gotpc
        ; the function body comes here
        函数体在这里
        mov     ebx,[ebp−4]
        动起来  Ebx [ ebp-4]
        mov     esp,ebp
        动起来  尤其是 ebp
```

118

```
        pop     ebp
        Pop ebp 流行音乐
        ret
        后悔
```

(For BSD, again, the symbol `_GLOBAL_OFFSET_TABLE` requires a second leading underscore.)
(对于 BSD，同样，符号 _ GLOBAL _ offset _ table 需要第二个前导下划线

The first two lines of this function are simply the standard C prologue to set up a stack frame, and the last three lines are standard C function epilogue. The third line, and the fourth to last line, save and restore the `EBX` register, because PIC shared libraries use this register to store the address of the GOT.
这个函数的前两行只是建立堆栈框架的标准 c 开场白，后三行是标准 c 函数的结束语。第三行，也是倒数第四行，保存和恢复 EBX 寄存器，因为 PIC 共享库使用这个寄存器来存储 GOT 的地址。

The interesting bit is the `CALL` instruction and the following two lines. The `CALL` and `POP` combination obtains the address of the label `.get_GOT`, without having to know in advance where the program was loaded (since the `CALL` instruction is encoded relative to the current position). The `ADD` instruction makes use of one of the special PIC relocation types: GOTPC relocation. With the `WRT ..gotpc` qualifier specified, the symbol referenced (here `_GLOBAL_OFFSET_TABLE_`, the special symbol assigned to the GOT) is given as an offset from the beginning of the section. (Actually, `ELF` encodes it as the offset from the operand field of the `ADD` instruction, but NASM simplifies this deliberately, so you do things the same way for both `ELF` and `BSD`.) So the instruction then *adds* the beginning of the section, to get the real address of the GOT, and subtracts the value of `.get_GOT` which it knows is in `EBX`. Therefore, by the time that instruction has finished, `EBX` contains the address of the GOT.
有趣的是 CALL 指令和以下两行。CALL 和 POP 组合获得标签的地址。Get _ got，而不必事先知道程序的加载位置(因为 CALL 指令是相对于当前位置编码的)。ADD 指令使用了一种特殊的 PIC 重定位类型: GOTPC 重定位。使用 WRT。.如果指定了 gotpc 限定符，则引用的符号(这里是 _ GLOBAL _ offset _ table _，分配给 GOT 的特殊符号)将作为从部分开头的偏移量给出。(实际上，ELF 将其编码为 ADD 指令的操作数字段的偏移量，但是 NASM 有意简化了这一点，所以对于 ELF 和 BSD 都采用相同的方法。) 所以，这个指令然后加上这一部分的开头，得到 GOT 的真实地址，然后减去。得到 _ got，它知道它在 EBX 中。因此，当指令完成时，EBX 包含 GOT 的地址。

If you didn’t follow that, don’t worry: it’s never necessary to obtain the address of the GOT by any other means, so you can put those three instructions into a macro and safely ignore them:
如果您没有遵循这一点，不要担心: 从来没有必要通过任何其他方式获取 GOT 的地址，因此您可以将这三条指令放入宏中并安全地忽略它们:

```
%macro  get_GOT 0
% 宏获取 _ got 0

        call    %%getgot
        打电话给我
  %%getgot:
  %% 得到:
        pop     ebx
        流行电子音乐
        add     ebx,_GLOBAL_OFFSET_TABLE_+$$-%%getgot wrt ..gotpc
        添加 ebx，_ global _ offset _ table _ + $$-% getgot wrt

%endmacro
% endmacro
```

9.2.2 Finding Your Local Data Items
9.2.2 查找本地数据项

Having got the GOT, you can then use it to obtain the addresses of your data items. Most variables will reside in the sections you have declared; they can be accessed using the `..gotoff` special `WRT` type. The way this works is like this:
得到 GOT 之后，您就可以使用它来获取数据项的地址。大多数变量将驻留在你声明的部分中; 它们可以使用。.特殊 WRT 类型。工作原理是这样的:

```
        lea     eax,[ebx+myvar wrt ..gotoff]
        Lea eax，[ ebx + myvar wrt. gotoff ]
```

The expression `myvar wrt ..gotoff` is calculated, when the shared library is linked, to be the offset to the local variable `myvar` from the beginning of the GOT. Therefore, adding it to `EBX` as above will place the real address of `myvar` in `EAX`.
表达式 myvar wrt。.当共享库被链接时，gotoff 被计算为从 GOT 开始到局部变量 myvar 的偏移量。因此，像上面那样添加到 EBX 将把 myvar 的真实地址放在 EAX 中。

If you declare variables as `GLOBAL` without specifying a size for them, they are shared between code modules in the library, but do not get exported from the library to the program that loaded it. They will still be in your ordinary data and BSS sections, so you can access them in the same way as local variables, using the above `..gotoff` mechanism.
如果将变量声明为 GLOBAL 而不指定其大小，则它们将在库中的代码模块之间共享，但不会从库导出到加载它的程序。它们仍然存在于你的普通数据和 BSS 部分，所以你可以像访问本地变量一样使用上面的方法访问它们。.Gooff 机制。

Note that due to a peculiarity of the way BSD `a.out` format handles this relocation type, there must be at least one non−local symbol in the same section as the address you're trying to access.
注意，由于 BSD. out 格式处理此重定位类型的方式的特殊性，在您试图访问的地址的同一节中必须至少有一个非本地符号。

9.2.3 Finding External and Common Data Items
9.2.3 查找外部和公共数据项

If your library needs to get at an external variable (external to the *library*, not just to one of the modules within it), you must use the `..got` type to get at it. The `..got` type, instead of giving you the offset from the GOT base to the variable, gives you the offset from the GOT base to a GOT *entry* containing the address of the variable. The linker will set up this GOT entry when it builds the library, and the dynamic linker will place the correct address in it at load time. So to obtain the address of an external variable `extvar` in `EAX`, you would code
如果库需要获取外部变量(库的外部变量，而不仅仅是库中的某个模块)，则必须使用。.获取类型。译者:。.GOT type，而不是给出从 GOT 基到变量的偏移量，而是给出从 GOT 基到包含变量地址的 GOT 条目的偏移量。链接器将在构建库时设置这个 GOT 条目，而动态链接器将在加载时在其中放置正确的地址。因此，为了获得 EAX 中一个外部变量 extvar 的地址，你需要编写代码

```
        mov     eax,[ebx+extvar wrt ..got]
        Mov eax，[ ebx + extvar wrt. . got ]
```

This loads the address of `extvar` out of an entry in the GOT. The linker, when it builds the shared library, collects together every relocation of type `..got`, and builds the GOT so as to ensure it has every necessary entry present.
这将从 GOT 中的一个条目中加载 extvar 的地址。链接器在构建共享库时，会收集所有类型的重定位。.然后构建 GOT，以确保它拥有所有必要的条目。

Common variables must also be accessed in this way.
公共变量也必须以这种方式访问。

9.2.4 Exporting Symbols to the Library User
9.2.4 向库用户导出符号

If you want to export symbols to the user of the library, you have to declare whether they are functions or data, and if they are data, you have to give the size of the data item. This is because the dynamic linker has to build procedure linkage table entries for any exported functions, and also moves exported data items away from the library's data section in which they were declared.
如果要向库的用户导出符号，则必须声明它们是函数还是数据，如果它们是数据，则必须给出数据项的大小。这是因为动态链接器必须为任何导出的函数构建过程链接表条目，并将导出的数据项从声明它们的库的数据部分移走。

So to export a function to users of the library, you must use
因此，要向库的用户导出函数，必须使用

```
global  func:function           ; declare it as a function
全局函数: 函数; 将其声明为函数

func:   push    ebp
Func: push ebp

        ; etc.
        等等。
```

And to export a data item such as an array, you would have to code
并且要导出数组等数据项，必须编写代码

```
global  array:data array.end-array      ; give the size too
全局数组: data array. end-array; 也给出大小

array:  resd    128
数组: resd 128
.end:
结束:
```

Be careful: If you export a variable to the library user, by declaring it as `GLOBAL` and supplying a size, the variable will end up living in the data section of the main program, rather than in your library's data section, where you declared it. So you will have to access your own global variable with the `..got` mechanism rather than `..gotoff`, as if it were external (which, effectively, it has become).
注意: 如果您将一个变量导出给库用户，通过将其声明为 GLOBAL 并提供一个大小，那么该变量最终将存在于主程序的数据部分，而不是您声明它的库的数据部分。所以你必须使用。.获取机制，而不是。.仿佛它是外部的(实际上，它已经变成了外部的)。

Equally, if you need to store the address of an exported global in one of your data sections, you can't do it by means of the standard sort of code:
同样，如果您需要将导出的全局地址存储在数据部分中，则不能使用标准类型的代码:

```
dataptr:        dd      global_data_item        ; WRONG
Dataptr: dd global _ data _ item; WRONG
```

NASM will interpret this code as an ordinary relocation, in which `global_data_item` is merely an offset from the beginning of the `.data` section (or whatever); so this reference will end up pointing at your data section instead of at the exported global which resides elsewhere.
NASM 将把这段代码解释为一个普通的重定位，其中全局 _ data _ 项仅仅是从。所以这个引用最终将指向你的数据部分，而不是其他地方的导出的全局。

Instead of the above code, then, you must write
代替上面的代码，然后，你必须写

```
dataptr:        dd      global_data_item wrt ..sym
Dataptr: dd global _ data _ item wrt. sym
```

which makes use of the special `WRT` type `..sym` to instruct NASM to search the symbol table for a particular symbol at that address, rather than just relocating by section base.
它使用了特殊的 WRT 类型。.Sym 指示 NASM 在符号表中搜索该地址的特定符号，而不仅仅是根据区域基础重新定位。

Either method will work for functions: referring to one of your functions by means of
这两种方法都适用于函数: 通过

```
funcptr:        dd      my_function
Funcptr: dd my _ function
```

will give the user the address of the code you wrote, whereas
将向用户提供您编写的代码的地址，而

```
funcptr:        dd      my_function wrt ..sym
Funcptr: dd my _ function wrt. sym
```

will give the address of the procedure linkage table for the function, which is where the calling program will *believe* the function lives. Either address is a valid way to call the function.
将给出函数的过程链接表的地址，调用程序将相信函数存在于此处。任何一个地址都是调用函数的有效方法。

9.2.5 Calling Procedures Outside the Library
9.2.5 在图书馆外呼叫程序

Calling procedures outside your shared library has to be done by means of a *procedure linkage table*, or PLT. The PLT is placed at a known offset from where the library is loaded, so the library code can make calls to the PLT in a position−independent way. Within the PLT there is code to jump to offsets contained in the GOT, so function calls to other shared libraries or to routines in the main program can be transparently passed off to their real destinations.
在共享库之外调用过程必须通过过程链接表或 PLT 来完成。PLT 被放置在与库加载位置相距的已知偏移量处，因此库代码可以以位置无关的方式对 PLT 进行调用。在 PLT 中有跳转到 GOT 中包含的偏移量的代码，因此对其他共享库或主程序中的例程的函数调用可以透明地传递到它们的真正目的地。

To call an external routine, you must use another special PIC relocation type, `WRT ..plt`. This is much easier than the GOT−based ones: you simply replace calls such as `CALL printf` with the PLT−relative version `CALL printf WRT ..plt`.
要调用外部例程，必须使用另一种特殊的 PIC 重定位类型 WRT。.译者:。这比基于 GOT 的方法要简单得多: 你只需要用 PLT 相对版本 CALL printf WRT 替换 CALL printf 之类的调用。.译注:。

9.2.6 Generating the Library File
9.2.6 生成库文件

Having written some code modules and assembled them to `.o` files, you then generate your shared library with a command such as
编写了一些代码模块并将它们组装到。O 文件，然后使用以下命令生成共享库，如

```
ld -shared -o library.so module1.o module2.o       # for ELF
Ld-shared-o library. so module1.o module2.o # for ELF
ld -Bshareable -o library.so module1.o module2.o   # for BSD
Ld-Bshareable-o library. so module1.o module2.o # for BSD
```

For ELF, if your shared library is going to reside in system directories such as `/usr/lib` or `/lib`, it is usually worth using the `-soname` flag to the linker, to store the final library file name, with a version number, into the library:
对于 ELF，如果您的共享库将驻留在/usr/lib 或/lib 之类的系统目录中，那么通常值得使用链接器的-soname 标志将最终的库文件名(带有版本号)存储到库中:

```
ld -shared -soname library.so.1 -o library.so.1.2 *.o
Id-shared-soname library 1-o library 1.2 * . o
```

You would then copy `library.so.1.2` into the library directory, and create `library.so.1` as a symbolic link to it.
然后将 library.so. 1.2 复制到 library 目录中，并创建 library.so. 1 作为指向该目录的符号链接。

121

122

Chapter 10: Mixing 16 and 32 Bit Code
第十章: 混合 16 和 32 位代码

This chapter tries to cover some of the issues, largely related to unusual forms of addressing and jump instructions, encountered when writing operating system code such as protected−mode initialisation routines, which require code that operates in mixed segment sizes, such as code in a 16−bit segment trying to modify data in a 32−bit one, or jumps between different−size segments.
本章试图涵盖在编写操作系统代码(如保护模式初始化例程)时遇到的一些问题，这些问题主要与寻址和跳转指令的不寻常形式有关，这些代码需要以混合段大小进行操作，如 16 位段中的代码试图修改 32 位段中的数据，或者在不同大小的段之间跳转。

10.1 Mixed−Size Jumps
10.1 混合大小跳跃

The most common form of mixed−size instruction is the one used when writing a 32−bit OS: having done your setup in 16−bit mode, such as loading the kernel, you then have to boot it by switching into protected mode and jumping to the 32−bit kernel start address. In a fully 32−bit OS, this tends to be the *only* mixed−size instruction you need, since everything before it can be done in pure 16−bit code, and everything after it can be pure 32−bit.
最常见的混合大小指令是在编写 32 位操作系统时使用的指令: 在 16 位模式下完成设置，比如加载内核，然后必须通过切换到保护模式并跳转到 32 位内核开始地址来引导它。在一个完全 32 位的操作系统中，这往往是您所需要的唯一混合大小的指令，因为之前的所有操作都可以用纯粹的 16 位代码完成，之后的所有操作都可以用纯粹的 32 位代码完成。

This jump must specify a 48−bit far address, since the target segment is a 32−bit one. However, it must be assembled in a 16−bit segment, so just coding, for example,
此跳转必须指定一个 48 位的远地址，因为目标段是 32 位的。然而，它必须组装在一个 16 位的段中，所以只需要编码，例如,

```
jmp     0x1234:0x56789ABC       ; wrong!
错误!
```

will not work, since the offset part of the address will be truncated to `0x9ABC` and the jump will be an ordinary 16−bit far one.
将无法工作，因为地址的偏移量部分将被截断为 0x9abc，跳转将是一个普通的 16 位远 1。

The Linux kernel setup code gets round the inability of `as86` to generate the required instruction by coding it manually, using `DB` instructions. NASM can go one better than that, by actually generating the right instruction itself. Here’s how to do it right:
Linux 内核设置代码通过使用 DB 指令手动编码来解决 as86 无法生成所需指令的问题。NASM 可以做得更好，通过实际生成正确的指令。下面是如何正确使用的方法:

```
jmp     dword 0x1234:0x56789ABC         ; right
Jmp dword 0x1234:0x56789ABC; right
```

The `DWORD` prefix (strictly speaking, it should come *after* the colon, since it is declaring the *offset* field to be a doubleword; but NASM will accept either form, since both are unambiguous) forces the offset part to be treated as far, in the assumption that you are deliberately writing a jump from a 16−bit segment to a 32−bit one.
DWORD 前缀(严格来说，它应该在冒号后面，因为它声明偏移字段是一个双字; 但是 NASM 将接受任何一种形式，因为两种形式都是明确的)强制将偏移部分处理得尽可能远，假设您是故意写一个从 16 位段到 32 位段的跳转。

You can do the reverse operation, jumping from a 32−bit segment to a 16−bit one, by means of the
可以执行相反的操作，从 32 位段跳转到 16 位段
`WORD` prefix:
`WORD` 前缀:

```
        jmp     word 0x8765:0x4321      ; 32 to 16 bit
        Jmp word 0x8765:0x4321; 32 到 16 位
```

If the `WORD` prefix is specified in 16−bit mode, or the `DWORD` prefix in 32−bit mode, they will be ignored, since each is explicitly forcing NASM into a mode it was in anyway.
如果 WORD 前缀是在 16 位模式下指定的，或者 DWORD 前缀是在 32 位模式下指定的，它们将被忽略，因为每个前缀都显式地迫使 NASM 进入它所处的模式。

10.2 Addressing Between Different−Size Segments
10.2 不同大小段之间的寻址

If your OS is mixed 16 and 32−bit, or if you are writing a DOS extender, you are likely to have to deal with some 16−bit segments and some 32−bit ones. At some point, you will probably end up writing code in a 16−bit segment which has to access data in a 32−bit segment, or vice versa.
如果您的操作系统是 16 位和 32 位混合的，或者如果您正在编写 DOS 扩展程序，您可能需要处理一些 16 位段和一些 32 位段。在某种程度上，您可能最终会在一个 16 位段中编写代码，而这个 16 位段必须访问一个 32 位段中的数据，反之亦然。

If the data you are trying to access in a 32−bit segment lies within the first 64K of the segment, you may be able to get away with using an ordinary 16−bit addressing operation for the purpose; but sooner or later, you will want to do 32−bit addressing from 16−bit mode.
如果您试图访问 32 位段中的数据位于该段的第一个 64k 内，则可以使用普通的 16 位寻址操作; 但是，您迟早会希望从 16 位模式进行 32 位寻址。

The easiest way to do this is to make sure you use a register for the address, since any effective address containing a 32−bit register is forced to be a 32−bit address. So you can do
最简单的方法是确保使用地址的寄存器，因为任何包含 32 位寄存器的有效地址都必须是 32 位地址。所以你可以这么做

```
        mov     eax,offset_into_32_bit_segment_specified_by_fs
        偏移到 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _
        mov     dword [fs:eax],0x11223344
        Mov dword [ fs: eax ] , 0 x11223344
```

This is fine, but slightly cumbersome (since it wastes an instruction and a register) if you already know the precise offset you are aiming at. The x86 architecture does allow 32−bit effective addresses to specify nothing but a 4−byte offset, so why shouldn't NASM be able to generate the best instruction for the purpose?
这很好，但是如果你已经知道目标的精确偏移量，这就有点麻烦了(因为这会浪费指令和寄存器)。X86 体系结构允许 32 位有效地址只指定 4 字节的偏移量，那么为什么 NASM 不能为此目的生成最好的指令呢？

It can. As in section 10.1, you need only prefix the address with the `DWORD` keyword, and it will be forced to be a 32−bit address:
可以的。如第 10.1 节所述，您只需在地址前加上 DWORD 关键字，它就必须是一个 32 位地址:

```
mov     dword [fs:dword my_offset],0x11223344
Mov dword [ fs: dword my _ offset ] , 0 x11223344
```

Also as in section 10.1, NASM is not fussy about whether the `DWORD` prefix comes before or after the segment override, so arguably a nicer−looking way to code the above instruction is
同样在第 10.1 节中，NASM 对于 DWORD 前缀是在段覆盖之前还是之后并不挑剔，因此可以说上述指令的一种更好看的编码方式是

```
mov     dword [dword fs:my_offset],0x11223344
Mov dword [ dword fs: my _ offset ] , 0 x11223344
```

Don't confuse the `DWORD` prefix *outside* the square brackets, which controls the size of the data stored at the address, with the one `inside` the square brackets which controls the length of the address itself. The two can quite easily be different:
不要混淆方括号外的 DWORD 前缀(它控制存储在地址中的数据的大小)和方括号内的 DWORD 前缀(它控制地址本身的长度)。这两者很容易区别开来:

```
mov     word [dword 0x12345678],0x9ABC
Mov word [ dword 0x12345678]0 x9ABC
```

This moves 16 bits of data to an address specified by a 32−bit offset.
这会将 16 位数据移动到由 32 位偏移量指定的地址。

You can also specify `WORD` or `DWORD` prefixes along with the `FAR` prefix to indirect far jumps or calls.
你也可以指定 WORD 或 DWORD 前缀以及 FAR 前缀来间接跳转或调用。
For example:
例如:

```
call    dword far [fs:word 0x4321]
Call dword far [ fs: word 0x4321]
```

This instruction contains an address specified by a 16−bit offset; it loads a 48−bit far pointer from that (16−bit segment and 32−bit offset), and calls that address.
此指令包含一个由 16 位偏移量指定的地址; 它从该地址(16 位段和 32 位偏移量)加载一个 48 位的远程指针，并调用该地址。

10.3 Other Mixed−Size Instructions
10.3 其他混合大小的指令

The other way you might want to access data might be using the string instructions (`LODSx`, `STOSx` and so on) or the `XLATB` instruction. These instructions, since they take no parameters, might seem to have no easy way to make them perform 32−bit addressing when assembled in a 16−bit segment.
另一种访问数据的方式可能是使用字符串指令(LODSx、 STOSx 等)或 XLATB 指令。这些指令，因为它们没有参数，似乎没有简单的方法使它们执行 32 位寻址时，组装在一个 16 位段。

This is the purpose of NASM's `a16`, `a32` and `a64` prefixes. If you are coding `LODSB` in a 16−bit segment but it is supposed to be accessing a string in a 32−bit segment, you should load the desired address into `ESI` and then code
这就是 NASM 的 a16，a32 和 a64 前缀的用途。如果您在 16 位段中编写 LODSB，但它应该访问 32 位段中的字符串，则应该将所需的地址加载到 ESI 中，然后编写代码

```
        a32     lodsb
        A32 lodsb
```

The prefix forces the addressing size to 32 bits, meaning that LODSB loads from [DS:ESI] instead of [DS:SI]. To access a string in a 16-bit segment when coding in a 32-bit one, the corresponding a16 prefix can be used.
前缀强制地址大小为 32 位，这意味着 LODSB 从[DS: ESI]而不是[DS: SI]加载。当编码为 32 位时，要访问 16 位段中的字符串，可以使用相应的 a16 前缀。

The a16, a32 and a64 prefixes can be applied to any instruction in NASM's instruction table, but most of them can generate all the useful forms without them. The prefixes are necessary only for instructions with implicit addressing: CMPSx, SCASx, LODSx, STOSx, MOVSx, INSx, OUTSx, and XLATB. Also, the various push and pop instructions (PUSHA and POPF as well as the more usual PUSH and POP) can accept a16, a32 or a64 prefixes to force a particular one of SP, ESP or RSP to be used as a stack pointer, in case the stack segment in use is a different size from the code segment.
A16、 a32 和 a64 前缀可以应用于 NASM 指令表中的任何指令，但是大多数前缀都可以不用它们生成所有有用的形式。这些前缀只对隐式寻址的指令是必需的: CMPSx，SCASx，LODSx，STOSx，MOVSx，INSx，OUTSx，和 XLATB。此外，各种 PUSH 和 POP 指令(PUSHA 和 POPF 以及更常见的 PUSH 和 POP)可以接受 a16、 a32 或 a64 前缀，以强制使用 SP、 ESP 或 RSP 中的某个特定前缀作为堆栈指针，以防所使用的堆栈段与代码段大小不同。

PUSH and POP, when applied to segment registers in 32-bit mode, also have the slightly odd behaviour that they push and pop 4 bytes at a time, of which the top two are ignored and the bottom two give the value of the segment register being manipulated. To force the 16-bit behaviour of segment-register push and pop instructions, you can use the operand-size prefix o16:
PUSH 和 POP 在应用于 32 位模式的段寄存器时，也有一些稍微奇怪的行为，它们一次推和弹出 4 个字节，其中前两个字节被忽略，后两个字节给出被操纵的段寄存器的值。为了强制执行段寄存器推送和弹出指令的 16 位行为，你可以使用操作数大小的前缀 o16:

```
                ss
                小
o16  push       学
O16  推         生
o16  push       ds
O16  推         D
```

124

This code saves a doubleword of stack space by fitting two segment registers into the space which would normally be consumed by pushing one.
这段代码通过将两个片段寄存器插入通常只需推送一个就可以消耗的空间，从而节省了一个双字堆栈空间。

(You can also use the `o32` prefix to force the 32−bit behaviour when in 16−bit mode, but this seems less useful.)
(在 16 位模式下，您也可以使用 o32 前缀强制执行 32 位行为，但这似乎没那么有用。)

125

126

Chapter 11: Writing 64−bit Code (Unix, Win64)
第十一章: 编写 64 位代码(Unix，Win64)

This chapter attempts to cover some of the common issues involved when writing 64−bit code, to run under Win64 or Unix. It covers how to write assembly code to interface with 64−bit C routines, and how to write position−independent code for shared libraries.
本章试图涵盖在 win64 或 Unix 下运行 64 位代码时所涉及的一些常见问题。它涵盖了如何编写汇编代码来接口 64 位 c 例程，以及如何编写位置无关的共享库代码。

All 64−bit code uses a flat memory model, since segmentation is not available in 64−bit mode. The one exception is the `FS` and `GS` registers, which still add their bases.
所有 64 位代码都使用平面内存模型，因为在 64 位模式下无法进行分段。一个例外是 FS 和 GS 寄存器，它们仍然添加它们的碱基。

Position independence in 64−bit mode is significantly simpler, since the processor supports `RIP`–relative addressing directly; see the `REL` keyword (section 3.3). On most 64−bit platforms, it is probably desirable to make that the default, using the directive `DEFAULT REL` (section 6.2).
64 位模式下的位置独立性非常简单，因为处理器直接支持 RIP 相对寻址; 参见 REL 关键字(第 3.3 节)。在大多数 64 位平台上，使用 DEFAULT REL 指令(第 6.2 节)使其成为默认设置可能是可取的。

64−bit programming is relatively similar to 32−bit programming, but of course pointers are 64 bits long; additionally, all existing platforms pass arguments in registers rather than on the stack. Furthermore, 64−bit platforms use SSE2 by default for floating point. Please see the ABI documentation for your platform.
64 位编程与 32 位编程相对类似，但当然指针长度为 64 位; 此外，所有现有平台都在寄存器而不是堆栈中传递参数。此外，64 位平台默认使用 sse2 作为浮点数。请参阅您的平台的 ABI 文档。

64−bit platforms differ in the sizes of the C/C++ fundamental datatypes, not just from 32−bit platforms but from each other. If a specific size data type is desired, it is probably best to use the types defined in the standard C header `<inttypes.h>`.
64 位平台在 c/c + + 基本数据类型的大小上有所不同，这不仅来自 32 位平台，而且来自其他平台。如果需要特定大小的数据类型，最好使用标准 c 头文件 < inttypes.h > 中定义的类型。

All known 64−bit platforms except some embedded platforms require that the stack is 16−byte aligned at the entry to a function. In order to enforce that, the stack pointer (`RSP`) needs to be aligned on an `odd` multiple of 8 bytes before the `CALL` instruction.
除了一些嵌入式平台外，所有已知的 64 位平台都要求堆栈在函数入口处对齐 16 字节。为了实现这一点，栈指针(RSP)需要在 CALL 指令之前以 8 字节的奇数倍对齐。

In 64−bit mode, the default instruction size is still 32 bits. When loading a value into a 32−bit register (but not an 8− or 16−bit register), the upper 32 bits of the corresponding 64−bit register are set to zero.
在 64 位模式下，默认的指令大小仍然是 32 位。当将值加载到 32 位寄存器(但不是 8 位或 16 位寄存器)时，相应的 64 位寄存器的上 32 位被设置为零。

11.1 Register Names in 64−bit Mode
11.164 位模式下的 Register Names

NASM uses the following names for general−purpose registers in 64−bit mode, for 8−, 16−, 32− and 64−bit references, respectively:
NASM 分别为 8 位、16 位、32 位和 64 位引用使用下列名称作为 64 位模式的通用寄存器:

```
AL/AH, CL/CH, DL/DH, BL/BH, SPL, BPL, SIL, DIL, R8B-R15B
AL/AH，CL/CH，DL/DH，BL/BH，SPL，BPL，SIL，DIL，R8B-R15B
AX, CX, DX, BX, SP, BP, SI, DI, R8W-R15W
AX，CX，DX，BX，SP，BP，SI，DI，R8W-R15W
EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, R8D-R15D
EAX，ECX，EDX，EBX，ESP，EBP，ESI，EDI，R8D-R15D
```

```
RAX, RCX, RDX, RBX, RSP, RBP, RSI, RDI, R8-R15
RAX，RCX，RDX，RBX，RSP，RBP，RSI，RDI，R8-R15
```

This is consistent with the AMD documentation and most other assemblers. The Intel documentation, however, uses the names `R8L-R15L` for 8−bit references to the higher registers. It is possible to use those names by definiting them as macros; similarly, if one wants to use numeric names for the low 8 registers, define them as macros. The standard macro package `altreg` (see section 5.1) can be used for this purpose.
这与 AMD 文档和大多数其他汇编程序一致。然而，Intel 文档使用名称 R8L-R15L 作为对更高寄存器的 8 位引用。可以通过将这些名称定义为宏来使用这些名称; 同样，如果希望对较低的 8 个寄存器使用数字名称，则将它们定义为宏。标准的宏包 altreg (见第 5.1 节)可以用于这个目的。

11.2 Immediates and Displacements in 64−bit Mode
11.264 位模式下的即时和位移

In 64−bit mode, immediates and displacements are generally only 32 bits wide. NASM will therefore truncate most displacements and immediates to 32 bits.
在 64 位模式下，即刻和位移通常只有 32 位宽。因此，NASM 将截断大部分位移，并立即到 32 位。

The only instruction which takes a full 64−bit immediate is:
唯一需要立即执行完整 64 位的指令是:

```
MOV reg64,imm64
MOV reg64, imm64
```

NASM will produce this instruction whenever the programmer uses `MOV` with an immediate into a 64−bit register. If this is not desirable, simply specify the equivalent 32−bit register, which will be automatically zero−extended by the processor, or specify the immediate as `DWORD`:
NASM 将产生这个指令，每当程序员使用 MOV 与一个立即进入 64 位寄存器。如果这是不可取的，只需指定等效的 32 位寄存器，它将自动零扩展处理器，或指定直接为 DWORD:

```
mov rax,foo                          ; 64−bit immediate
Mov rax foo                            64 位 马上
mov rax,qword foo               ; (identical)
Mov rax，qword foo              (相同的)
mov
动
起    eax,foo                          32−bit immediate, zero−extended
来    Eax foo                    ;      32 位 立即，零延长
mov
动
起    rax,dword foo                    32−bit immediate, sign−extended
来    Rax，dword foo             ;      32 位 立即，标志延伸
```

The length of these instructions are 10, 5 and 7 bytes, respectively.
这些指令的长度分别为 10、5 和 7 字节。

If optimization is enabled and NASM can determine at assembly time that a shorter instruction will suffice, the shorter instruction will be emitted unless of course `STRICT QWORD` or `STRICT DWORD` is specified (see section 3.7):
如果启用了优化，并且 NASM 可以在装配时确定较短的指令就足够了，那么将发出较短的指令，除非当然指定 STRICT QWORD 或 STRICT DWORD (见第 3.7 节) :

```
                                                  as
mov rax,1                       ; Assembles       作  "mov eax,1" (5 bytes)
Mov rax 1                       ; 装配            为  “ mov eax，1”(5 字节)
mov rax,strict qword 1  ; Full 10−byte                instruction
Mov rax，严格 qword 1; Full 10 字节                   指示
mov rax,strict dword 1          ; 7−byte instruction
Mov rax，严格的 dword 1         ; 7 字节指令
mov
动
起    rax,symbol                  10 bytes,      not known at assembly time
来    Rax 符号               ;    10 字节,       在集合时不知道
lea   rax,[rel symbol]            7 bytes, usually preferred by the ABI
:     Rax，[ rel 符号]       ;    7 字节，通常由 ABI 首选
```

Note that `lea rax,[rel symbol]` is position−independent, whereas `mov rax,symbol` is not. Most ABIs prefer or even require position−independent code in 64−bit mode. However, the `MOV` instruction is able to reference a symbol anywhere in the 64−bit address space, whereas `LEA` is only able to access a symbol within within 2 GB of the instruction itself (see below.)
注意 lea rax [rel 符号]是位置无关的，而 mov rax 不是。大多数 abi 更喜欢或者甚至需要 64 位模式下的位置无关代码。然而，MOV 指令能够在 64 位地址空间的任何位置引用符号，而 LEA 只能访问指令本身 2gb 内的符号(见下文)

The only instructions which take a full 64−bit *displacement* is loading or storing, using `MOV`, `AL`, `AX`, `EAX` or `RAX` (but no other registers) to an absolute 64−bit address. Since this is a relatively rarely used instruction (64−bit code generally uses relative addressing), the programmer has to explicitly declare the displacement size as `ABS QWORD`:
唯一需要完整 64 位位移的指令是加载或存储，使用 MOV、 AL、 AX、 EAX 或 RAX (但不包括其他寄存器)到绝对 64 位地址。由于这是一个相对较少使用的指令(64 位代码通常使用相对寻址)，程序员必须显式声明位移大小为 ABS QWORD:

```
default abs
默认腹肌

mov eax,[foo]                     ; 32−bit absolute disp, sign−extended
```

```
动起来                          32 位绝对损耗，符号扩展
mov eax,[a32 foo]               32−bit absolute disp, zero−extended
移动，移动，移动，移动        ; 32 位绝对损耗，零扩展
mov eax,[qword foo]             64−bit absolute disp
[英语]                        ; 64 位绝对损耗
default rel
默认相对数
mov eax,[foo]                   32−bit relative disp
动起来                        ; 32 位相对差
mov eax,[a32 foo]               d:o, address truncated to 32 bits(!)
移动，移动，移动，移动        ; D: o，地址截断为 32 位(!)
mov eax,[qword foo]             error
[英语]                        ; 错误
mov eax,[abs qword foo] ;       64−bit absolute disp
电影，电影                      64 位绝对损耗
```

A sign−extended absolute displacement can access from –2 GB to +2 GB; a zero−extended absolute displacement can access from 0 to 4 GB.
符号扩展的绝对位移可以访问 -2gb 到 + 2gb; 零扩展的绝对位移可以访问 0 到 4gb。

11.3 Interfacing to 64−bit C Programs (Unix)
11.364 位 c 程序的接口(Unix)

On Unix, the 64−bit ABI as well as the x32 ABI (32−bit ABI with the CPU in 64−bit mode) is defined by the documents at:
在 Unix 上，64 位 ABI 和 x32 ABI (CPU 处于 64 位模式的 32 位 ABI)由以下文档定义:

```
http://www.nasm.us/abi/unix64
Http://www.nasm.us/abi/unix64
```

Although written for AT&T−syntax assembly, the concepts apply equally well for NASM−style assembly.
虽然是为 at & t 语法汇编编写的，但这些概念同样适用于 NASM 风格的汇编。
What follows is a simplified summary.
下面是一个简单的总结。

The first six integer arguments (from the left) are passed in `RDI`, `RSI`, `RDX`, `RCX`, `R8`, and `R9`, in that order. Additional integer arguments are passed on the stack. These registers, plus `RAX`, `R10` and `R11` are destroyed by function calls, and thus are available for use by the function without saving.
前六个整数参数(从左开始)以 RDI、 RSI、 RDX、 RCX、 r8 和 r9 的顺序传递。额外的整数参数在堆栈上传递。这些寄存器，加上 RAX，r10 和 r11 被函数调用销毁，因此可供函数使用而无需保存。

Integer return values are passed in `RAX` and `RDX`, in that order.
Integer 返回值按照 RAX 和 RDX 的顺序传递。

Floating point is done using SSE registers, except for `long double`, which is 80 bits (`TWORD`) on most platforms (Android is one exception; there `long double` is 64 bits and treated the same as `double`.) Floating−point arguments are passed in `XMM0` to `XMM7`; return is `XMM0` and `XMM1`. `long double` are passed on the stack, and returned in `ST0` and `ST1`.
浮点是使用 SSE 寄存器完成的，除了 long double，在大多数平台上是 80 位(TWORD)(Android 是一个例外; long double 是 64 位，和 double 一样处理)浮点参数在 xmm0 中传递给 XMM7; 返回值是 xmm0 和 XMM1。长双精度函数在堆栈上传递，并在 st0 和 st1 中返回。

All SSE and x87 registers are destroyed by function calls.
所有 SSE 和 x87 寄存器都被函数调用销毁。

On 64−bit Unix, `long` is 64 bits.
在 64 位 Unix 上，long 等于 64 位。

Integer and SSE register arguments are counted separately, so for the case of
Integer 和 SSE 寄存器参数分别计数，因此对于

```
void foo(long a, double b, int c)
Void foo (long a，double b，int c)
```

`a` is passed in `RDI`, `b` in `XMM0`, and `c` in `ESI`.
A 在 RDI 中传递，b 在 xmm0 中传递，c 在 ESI 中传递。

11.4 Interfacing to 64−bit C Programs (Win64)
11.4 与 64 位 c 程序的接口(Win64)

The Win64 ABI is described by the document at:
Win64abi 的描述文件如下:

`http://www.nasm.us/abi/win64`
`Http://www.nasm.us/abi/win64`

What follows is a simplified summary.
下面是一个简单的总结。

The first four integer arguments are passed in `RCX`, `RDX`, `R8` and `R9`, in that order. Additional integer arguments are passed on the stack. These registers, plus `RAX`, `R10` and `R11` are destroyed by function calls, and thus are available for use by the function without saving.
前四个整数参数以 RCX，RDX，r8 和 r9 的顺序传递。额外的整数参数在堆栈上传递。这些寄存器，加上 RAX，r10 和 r11 被函数调用销毁，因此可供函数使用而无需保存。

Integer return values are passed in `RAX` only.
整数返回值仅用 RAX 传递。

Floating point is done using SSE registers, except for `long double`. Floating−point arguments are passed in `XMM0` to `XMM3`; return is `XMM0` only.
浮点是使用 SSE 寄存器完成的，除了长双精度浮点。浮点参数在 xmm0 中传递给 XMM3; 返回值仅为 XMM0。

On Win64, `long` is 32 bits; `long long` or `_int64` is 64 bits.
在 win64 上，long 是 32 位; long long 或 _int64 是 64 位。

Integer and SSE register arguments are counted together, so for the case of
Integer 和 SSE 寄存器参数一起计数，因此对于

```
void foo(long long a, double b, int c)
Void foo (long long a，double b，int c)
```

`a` is passed in `RCX`, `b` in `XMM1`, and `c` in `R8D`.
A 在 RCX 中传递，b 在 xmm1 中传递，c 在 R8D 中传递。

129

130

Chapter 12: Troubleshooting
第十二章: 故障排除

This chapter describes some of the common problems that users have been known to encounter with NASM, and answers them. If you think you have found a bug in NASM, please see section E.2.
本章描述了用户在使用 NASM 时遇到的一些常见问题，并给出了相应的解决方案。如果你认为你在 NASM 中发现了一个 bug，请参阅 e. 2 节。

12.1 Common Problems
12.1 常见问题

12.1.1 NASM Generates Inefficient Code
12.1.1 NASM 生成低效代码

We sometimes get ‘bug’ reports about NASM generating inefficient, or even ‘wrong’, code on instructions such as `ADD ESP,8`. This is a deliberate design feature, connected to predictability of output: NASM, on seeing `ADD ESP,8`, will generate the form of the instruction which leaves room for a 32−bit offset. You need to code `ADD ESP,BYTE 8` if you want the space−efficient form of the instruction. This isn’t a bug, it’s user error: if you prefer to have NASM produce the more efficient code automatically enable optimization with the `-O` option (see section 2.1.23).
我们有时会收到关于 NASM 生成低效甚至是错误代码的“ bug”报告，比如 ADD ESP，8。这是一个经过深思熟虑的设计特性，与输出的可预测性有关: NASM，在看到 ADD ESP，8 时，将生成指令的形式，为 32 位偏移留下空间。如果你想要节省空间的指令形式，你需要编写 ADD ESP，BYTE 8。这不是一个 bug，而是用户的错误: 如果您希望让 NASM 生成更高效的代码，则使用 -o 选项自动启用优化(参见第 2.1.23 节)。

12.1.2 My Jumps are Out of Range
12.1.2 我的跳跃超出范围

Similarly, people complain that when they issue conditional jumps (which are `SHORT` by default) that try to jump too far, NASM reports ‘short jump out of range’ instead of making the jumps longer.
同样，人们抱怨说，当他们发布条件跳跃(默认为 SHORT) ，试图跳得太远时，NASM 报告说“跳出范围的短距离跳跃”，而不是让跳跃变长。

This, again, is partly a predictability issue, but in fact has a more practical reason as well. NASM has no means of being told what type of processor the code it is generating will be run on; so it cannot decide for itself that it should generate `Jcc NEAR` type instructions, because it doesn’t know that it’s working for a 386 or above. Alternatively, it could replace the out−of−range short `JNE` instruction with a very short `JE` instruction that jumps over a `JMP NEAR`; this is a sensible solution for processors below a 386, but hardly efficient on processors which have good branch prediction *and* could have used `JNE NEAR` instead. So, once again, it’s up to the user, not the assembler, to decide what instructions should be generated. See section 2.1.23.
同样，这在一定程度上是一个可预测的问题，但事实上也有一个更实际的原因。NASM 没有办法被告知它正在生成的代码将在哪种类型的处理器上运行; 因此它不能自己决定是否应该生成 Jcc NEAR 类型指令，因为它不知道它正在为 386 或更高版本工作。或者，它可以用一个跳过 JMP NEAR 的非常短的 JE 指令来代替超出范围的短 JNE 指令; 对于低于 386 的处理器来说，这是一个合理的解决方案，但是对于具有良好的分支预测并且可以使用 JNE NEAR 来代替的处理器来说，效率很低。所以，再一次，这是由用户，而不是汇编程序，来决定什么指令应该生成。参见第 2.1.23 节。

12.1.3 `ORG` Doesn’t Work
12.1.3 ORG 不起作用

People writing boot sector programs in the `bin` format often complain that `ORG` doesn’t work the way they’d like: in order to place the `0xAA55` signature word at the end of a 512−byte boot sector, people who are used to MASM tend to code

使用 bin 格式编写引导扇区程序的人经常抱怨 ORG 没有按照他们希望的方式工作: 为了将 0xaa55 签名字放在 512 字节的引导扇区的末尾，习惯于 MASM 的人倾向于编写代码

```
ORG 0
ORG 0

; some boot sector code
; 一些启动扇区代码

ORG 510
ORG 510
DW 0xAA55
DW 0xAA55
```

This is not the intended use of the `ORG` directive in NASM, and will not work. The correct way to solve this problem in NASM is to use the `TIMES` directive, like this:
这不是在 NASM 中使用 ORG 指令的目的，也不会起作用。在 NASM 中解决这个问题的正确方法是使用 TIMES 指令，像这样:

```
ORG 0
ORG 0

; some boot sector code
; 一些启动扇区代码

TIMES 510-($-$$) DB 0
TIMES 510-($- $$) DB 0
DW 0xAA55
DW 0xAA55
```

The `TIMES` directive will insert exactly enough zero bytes into the output to move the assembly point up to 510. This method also has the advantage that if you accidentally fill your boot sector too full, NASM will catch the problem at assembly time and report it, so you won't end up with a boot sector that you have to disassemble to find out what's wrong with it.
TIMES 指令将在输出中插入足够多的零字节，将汇编点移动到 510。这种方法还有一个优点，即如果您意外地将引导扇区填得太满，NASM 将在装配时捕获问题并报告它，这样您就不会最终得到一个必须进行反汇编才能找出问题所在的引导扇区。

131

12.1.4 TIMES Doesn’t Work

12.1.4 TIMES doesn’t Work 12.1.4 次不起作用

The other common problem with the above code is people who write the TIMES line as
上述代码的另一个常见问题是将 TIMES 行编写为

```
        TIMES 510-$ DB 0
        TIMES 510-$DB 0
```

by reasoning that $ should be a pure number, just like 510, so the difference between them is also a pure number and can happily be fed to TIMES.
通过推理 $应该是一个纯数，就像 510 一样，所以它们之间的差异也是一个纯数，可以愉快地提供给 TIMES。

NASM is a *modular* assembler: the various component parts are designed to be easily separable for re−use, so they don’t exchange information unnecessarily. In consequence, the bin output format, even though it has been told by the ORG directive that the .text section should start at 0, does not pass that information back to the expression evaluator. So from the evaluator’s point of view, $ isn’t a pure number: it’s an offset from a section base. Therefore the difference between $ and 510 is also not a pure number, but involves a section base. Values involving section bases cannot be passed as arguments to TIMES.
NASM 是一个模块化的汇编程序: 不同的组成部分被设计成可以很容易地被重复使用，所以他们不会交换不必要的信息。因此，仓库输出格式，即使已经被 ORG 指令告知。文本部分应该从 0 开始，但不会将该信息传递回表达式计算器。因此，从计算器的角度来看，$不是一个纯数: 它是一个来自节基的偏移量。因此 $和 510 之间的差异也不是一个纯数，而是一个区间基数。含有区间基数的值不能作为参数传递给 TIMES。

The solution, as in the previous section, is to code the TIMES line in the form
与前一节一样，解决方案是在表单中对 TIMES 行进行编码

```
        TIMES 510-($-$$) DB 0
        TIMES 510-($- $$) DB 0
```

in which $ and $$ are offsets from the same section base, and so their difference is a pure number. This will solve the problem and generate sensible code.
其中 $和 $是同一节基的偏移量，因此它们之间的差是一个纯数。这样可以解决问题并生成合理的代码。

132

Appendix A: Ndisasm
附录 a: Ndisasm

The Netwide Disassembler, NDISASM
Netwide 反汇编器，NDISASM

A.1 Introduction
答. 1 引言

The Netwide Disassembler is a small companion program to the Netwide Assembler, NASM. It seemed a shame to have an x86 assembler, complete with a full instruction table, and not make as much use of it as possible, so here's a disassembler which shares the instruction table (and some other bits of code) with NASM.
Netwide 反汇编程序是 Netwide 汇编程序 NASM 的一个小型伴侣程序。有一个 x86 汇编程序，完整的指令表，却没有尽可能多地使用它，这似乎是一种耻辱，所以这里有一个反汇编程序，它与 NASM 共享指令表(和其他一些比特的代码)。

The Netwide Disassembler does nothing except to produce disassemblies of *binary* source files. NDISASM does not have any understanding of object file formats, like `objdump`, and it will not understand `DOS .EXE` files like `debug` will. It just disassembles.
Netwide 反汇编程序除了生成二进制源文件的反汇编之外什么也不做。NDISASM 不理解对象文件格式，比如 objdump，它也不理解 DOS。像 debug 这样的 EXE 文件。它只是反汇编。

A.2 Running NDISASM
A. 2 运行 NDISASM

To disassemble a file, you will typically use a command of the form
要反汇编文件，通常需要使用表单的命令

```
ndisasm -b {16|32|64} filename
Ndisasm-b {16 | 32 | 64}文件名
```

NDISASM can disassemble 16−, 32− or 64−bit code equally easily, provided of course that you remember to specify which it is to work with. If no `-b` switch is present, NDISASM works in 16−bit mode by default. The `-u` switch (for USE32) also invokes 32−bit mode.
NDISASM 可以同样容易地反汇编 16 位、32 位或 64 位代码，当然前提是您记得指定要使用哪些代码。如果没有 b 开关，NDISASM 默认工作在 16 位模式。U 开关(对于 USE32)也调用 32 位模式。

Two more command line options are `-r` which reports the version number of NDISASM you are running, and `-h` which gives a short summary of command line options.
另外两个命令行选项是 -r，它报告您正在运行的 NDISASM 的版本号，以及 -h，它给出命令行选项的简短摘要。

A.2.1 COM Files: Specifying an Origin
A. 2.1 COM 文件: 指定原点

To disassemble a `DOS .COM` file correctly, a disassembler must assume that the first instruction in the file is loaded at address `0x100`, rather than at zero. NDISASM, which assumes by default that any file you give it is loaded at zero, will therefore need to be informed of this.
反汇编 DOS。正确地使用 COM 文件，反汇编程序必须假设文件中的第一条指令是在 0x100 的地址加载的，而不是零。NDISASM，在默认情况下假设你给它的任何文件都是零加载的，因此需要通知它。

The `-o` option allows you to declare a different origin for the file you are disassembling. Its argument may be expressed in any of the NASM numeric formats: decimal by default, if it begins with '`$`' or '`0x`' or ends in '`H`' it's `hex`, if it ends in '`Q`' it's `octal`, and if it ends in '`B`' it's `binary`.

O 选项允许您为正在反汇编的文件声明一个不同的原点。它的参数可以用任何一种 NASM 数值格式表示: 默认情况下是十进制，如果它以“ $”或“0x”开头或以“ h”结尾，它就是十六进制，如果它以“ q”结尾，它就是八进制，如果它以“ b”结尾，它就是二进制。

Hence, to disassemble a `.COM` file:
因此，要反汇编一个.COM 文件:

```
        ndisasm -o100h filename.com
        Ndisasm-o100h filename. com
```

will do the trick.
会成功的。

A.2.2 Code Following Data: Synchronisation
A. 2.2 代码跟随数据: 同步

Suppose you are disassembling a file which contains some data which isn’t machine code, and *then* contains some machine code. NDISASM will faithfully plough through the data section, producing machine instructions wherever it can (although most of them will look bizarre, and some may have unusual prefixes, e.g. ‘`FS OR AX,0x240A`’), and generating ‘DB’ instructions ever so often if it’s totally stumped. Then it will reach the code section.
假设你正在反汇编一个文件，其中包含一些非机器代码的数据，然后又包含一些机器代码。NDISASM 将忠实地研究数据部分，尽可能地生成机器指令(尽管它们中的大多数看起来很奇怪，有些可能有不寻常的前缀，例如‘ FS OR AX，0 x240A’) ，如果它完全被难住了，则会经常生成‘ DB’指令。然后它会到达代码部分。

Supposing NDISASM has just finished generating a strange machine instruction from part of the data section, and its file position is now one byte *before* the beginning of the code section. It’s entirely possible that another spurious instruction will get generated, starting with the final byte of the data section, and then the correct first instruction in the code section will not be seen because the starting point skipped over it. This isn’t really ideal.
假设 NDISASM 刚刚从数据部分生成了一个奇怪的机器指令，它的文件位置现在是代码部分开始前的一个字节。完全有可能会生成另一条虚假指令，从数据部分的最后一个字节开始，然后代码部分中正确的第一条指令将不会被看到，因为起始点跳过了它。这并不是很理想。

To avoid this, you can specify a ‘synchronisation’ point, or indeed as many synchronisation points as you like (although NDISASM can only handle 2147483647 sync points internally). The definition of a sync point is this: NDISASM guarantees to hit sync points exactly during disassembly. If it is thinking about generating an instruction which would cause it to jump over a sync point, it will discard that instruction and output a ‘db’ instead. So it *will* start disassembly exactly from the sync point, and so you *will* see all the instructions in your code section.
为了避免这种情况，您可以指定一个“同步”点，或者您喜欢的任意多个同步点(尽管 NDISASM 内部只能处理 2147483647 个同步点)。同步点的定义是这样的: NDISASM 保证在拆卸过程中完全达到同步点。如果它正在考虑生成一条指令，这将导致它跳过一个同步点，它将丢弃该指令并输出一个‘ db’代替。所以它会从同步点开始准确地反汇编，这样你就可以在代码部分看到所有的指令。

Sync points are specified using the -s option: they are measured in terms of the program origin, not the file position. So if you want to synchronize after 32 bytes of a .COM file, you would have to do
同步点是使用 -s 选项指定的: 它们是根据程序原点而不是文件位置来衡量的。因此，如果你想在 32 个字节之后进行同步。COM 文件，你必须这样做

```
        ndisasm -o100h -s120h file.com
        Ndisasm-o100h-s120h file. com
```

rather than
而不是

```
        ndisasm -o100h -s20h file.com
        Ndisasm-o100h-s20h file. com
```

As stated above, you can specify multiple sync markers if you need to, just by repeating the -s option.
如上所述，你可以指定多个同步标记，如果你需要的话，只需要重复-s 选项。

A.2.3 Mixed Code and Data: Automatic (Intelligent) Synchronisation
A. 2.3 混合代码和数据: 自动(智能)同步

Suppose you are disassembling the boot sector of a DOS floppy (maybe it has a virus, and you need to understand the virus so that you know what kinds of damage it might have done you). Typically, this will contain a JMP instruction, then some data, then the rest of the code. So there is a very good chance of NDISASM being *misaligned* when the data ends and the code begins. Hence a sync point is needed.
假设您正在分解 DOS 软盘的引导扇区(也许它有病毒，您需要了解病毒，以便知道它可能对您造成了什么样的损害)。通常，这会包含一个 JMP 指令，然后是一些数据，最后是剩下的代码。因此，当数据结束，代码开始时，NDISASM 很有可能被错误对齐。因此需要一个同步点。

On the other hand, why should you have to specify the sync point manually? What you’d do in order to find where the sync point would be, surely, would be to read the JMP instruction, and then to use its target address as a sync point. So can NDISASM do that for you?
另一方面，为什么必须手动指定同步点？为了找到同步点的位置，您需要做的肯定是读取 JMP 指令，然后使用它的目标地址作为同步点。那么 NDISASM 能为你做到这一点吗？

The answer, of course, is yes: using either of the synonymous switches -a (for automatic sync) or -i (for intelligent sync) will enable auto-sync mode. Auto−sync mode automatically generates a sync point for any forward−referring PC−relative jump or call instruction that NDISASM encounters. (Since NDISASM is one−pass, if it encounters a PC−relative jump whose target has already been processed, there isn’t much it can do about it...)
当然，答案是肯定的: 使用两种同义开关—— a (用于自动同步)或 -i (用于智能同步)将启用自动同步模式。自动同步模式会自动为 NDISASM 遇到的任何前向引用 PC 相对跳转或调用指令生成一个同步点。(由于 NDISASM 是一次性的，如果它遇到一个目标已经被处理过的 PC 相对跳转，那么它也无能为力... ...)

Only PC−relative jumps are processed, since an absolute jump is either through a register (in which case NDISASM doesn’t know what the register contains) or involves a segment address (in which case the target code isn’t in the same segment that NDISASM is working in, and so the sync point can’t be placed anywhere useful).

只处理 PC 相对跳转，因为绝对跳转要么是通过寄存器(在这种情况下，NDISASM 不知道寄存器包含什么)，要么涉及段地址(在这种情况下，目标代码不在 NDISASM 工作的相同段中，因此同步点不能放在任何有用的地方)。

For some kinds of file, this mechanism will automatically put sync points in all the right places, and save you from having to place any sync points manually. However, it should be stressed that auto−sync mode is *not* guaranteed to catch all the sync points, and you may still have to place some manually.
对于某些类型的文件，这种机制会自动将同步点放置在所有正确的位置，从而使您不必手动放置任何同步点。但是，应该强调的是，自动同步模式不能保证捕获所有的同步点，你可能仍然需要手动放置一些。

Auto−sync mode doesn't prevent you from declaring manual sync points: it just adds automatically generated ones to the ones you provide. It's perfectly feasible to specify `−i` *and* some `−s` options.
自动同步模式并不妨碍您声明手动同步点: 它只是将自动生成的点添加到您提供的点。指定 -i 和一些 -s 选项是完全可行的。

Another caveat with auto−sync mode is that if, by some unpleasant fluke, something in your data section should disassemble to a PC−relative call or jump instruction, NDISASM may obediently place a sync point in a totally random place, for example in the middle of one of the instructions in your code section. So you may end up with a wrong disassembly even if you use auto−sync. Again, there isn't much I can do about this. If you have problems, you'll have to use manual sync points, or use the `−k` option (documented below) to suppress disassembly of the data area.
自动同步模式的另一个注意事项是，如果由于某种不愉快的侥幸，数据部分中的某些内容应该反汇编为 PC 相关调用或跳转指令，NDISASM 可能会顺从地在一个完全随机的地方放置一个同步点，例如在代码部分的一个指令中间。所以即使你使用自动同步，你也可能最终会出现错误的反汇编。同样，我对此也无能为力。如果您有问题，您将不得不使用手动同步点，或者使用 -k 选项(见下文)来抑制数据区域的反汇编。

A.2.4 Other Options
A. 2.4 其他选择

The `−e` option skips a header on the file, by ignoring the first N bytes. This means that the header is *not* counted towards the disassembly offset: if you give `−e10 −o10`, disassembly will start at byte 10 in the file, and this will be given offset 10, not 20.
E 选项通过忽略前 n 个字节跳过文件的头部。这意味着头部不计入反汇编偏移量: 如果给出 -e10-o10，反汇编将从文件中的字节 10 开始，并且将给出偏移量 10，而不是 20。

The `-k` option is provided with two comma−separated numeric arguments, the first of which is an assembly offset and the second is a number of bytes to skip. This *will* count the skipped bytes towards the assembly offset: its use is to suppress disassembly of a data section which wouldn’t contain anything you wanted to see anyway.
K 选项提供了两个逗号分隔的数值参数，第一个是程序集偏移量，第二个是要跳过的字节数。这将把跳过的字节计入程序集偏移量: 它的用途是抑制数据部分的反汇编，该部分不会包含任何您想要看到的内容。

135

136

Appendix B: Instruction List
附录 b: 指令表

B.1 Introduction
B. 1 引言

The following sections show the instructions which NASM currently supports. For each instruction, there is a separate entry for each supported addressing mode. The third column shows the processor type in which the instruction was introduced and, when appropriate, one or more usage flags.
以下部分显示 NASM 目前支持的说明。对于每条指令，每个支持的寻址模式都有一个单独的条目。第三列显示了引入指令的处理器类型，以及在适当的时候，一个或多个使用标志。

B.1.1 Special instructions (pseudo−ops)
1.1 特殊指令(伪操作)

```
DB
数据库
DW
DW
DD
弟弟
DQ
DQ
DT
DT
DO
DO
DY
DY
DZ
DZ
RESB                imm                        8086
8086
RESW
再生资源
RESD
RESD
RESQ
RESQ
REST
休息
RESO
雷索(RESO)
RESY
RESY
RESZ
雷斯
INCBIN
INCBIN
```

B.1.2 Conventional instructions
B. 1.2 常规说明

AAA	8086,NOLONG
汽车协会	8086，

		NOLONG
		8086,NOLONG
AAD		8086，
AAD		NOLONG
		8086,NOLONG
AAD	imm	8086，
AAD	是的	NOLONG
		8086,NOLONG
AAM		8086，
AAM		NOLONG
		8086,NOLONG
AAM	imm	8086，
AAM	是的	NOLONG
AAS		8086,NOLONG
原子吸收光		8086，
谱仪		NOLONG
ADC	mem,reg8	8086,LOCK
ADC	Mem reg8	8086，LOCK
ADC	reg8,reg8	
ADC	Reg8，reg8	8086
ADC	mem,reg16	8086,LOCK
ADC	Mem，reg16	8086，LOCK
ADC	reg16,reg16	
ADC	Reg16，reg16	8086
ADC	mem,reg32	386,LOCK
ADC	Mem reg32	386，LOCK
ADC	reg32,reg32	
ADC	Reg32，reg32	386
ADC	mem,reg64	X64,LOCK
ADC	Mem，reg64	X64，锁定
ADC	reg64,reg64	X64
ADC	Reg64，reg64	X64
ADC	reg8,mem	
ADC	Reg8，mem	8086
ADC	reg8,reg8	
ADC	Reg8，reg8	8086
ADC	reg16,mem	
ADC	Reg16，mem	8086
ADC	reg16,reg16	
ADC	Reg16，reg16	8086
ADC	reg32,mem	
ADC	Reg32，mem	386
ADC	reg32,reg32	
ADC	Reg32，reg32	386
ADC	reg64,mem	X64
ADC	Reg64，mem	X64
ADC	reg64,reg64	X64
ADC	Reg64，reg64	X64
ADC	rm16,imm8	8086,LOCK
ADC	Rm16，imm8	8086，LOCK

137

ADC	rm32,imm8	386,LOCK
ADC	Rm32，imm8	386，LOCK
ADC	rm64,imm8	X64,LOCK
ADC	Rm64，imm8	X64，锁定
ADC	reg_al,imm	
ADC	我叫 reg _ al	8086
ADC	reg_ax,sbyteword	8086,ND
ADC	Reg _ ax，字节词	8086，ND
ADC	reg_ax,imm	
ADC	Reg _ ax，imm	8086
ADC	reg_eax,sbytedword	386,ND
ADC	Reg _ eax，bytedword	新界 386 号
ADC	reg_eax,imm	
ADC	注意，注意	386
ADC	reg_rax,sbytedword	X64,ND
ADC	Reg _ rax，字节词	X64，ND
ADC	reg_rax,imm	X64
ADC	Reg _ rax，imm	X64
ADC	rm8,imm	8086,LOCK
ADC	Rm8，imm	8086，LOCK
ADC	rm16,sbyteword	8086,LOCK,ND
ADC	Rm16，字节词	新界洛克 8086 号
ADC	rm16,imm	8086,LOCK
ADC	16 元	8086，LOCK
ADC	rm32,sbytedword	386,LOCK,ND
ADC	Rm32，字节词	新界洛克 386 号
ADC	rm32,imm	386,LOCK
ADC	Rm32，imm	386，LOCK
ADC	rm64,sbytedword	X64,LOCK,ND
ADC	Rm64，字节词	X64，LOCK，ND
ADC	rm64,imm	X64,LOCK
ADC	64 元	X64，锁定
ADC	mem,imm8	8086,LOCK,ND
ADC	Mem，imm8	新界洛克 8086 号
ADC	mem,sbyteword16	8086,LOCK,ND
ADC	Mem，byteword16	新界洛克 8086 号
ADC	mem,imm16	8086,LOCK
ADC	我是 im16	8086，LOCK
ADC	mem,sbytedword32	386,LOCK,ND
ADC	Mem，sbytedword32	新界洛克 386 号
ADC	mem,imm32	386,LOCK
ADC	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
ADC	rm8,imm	8086，LOCK，ND，
ADC	Rm8，imm	NOLONG
ADD	mem,reg8	8086,LOCK
ADD 注释	Mem reg8	8086，LOCK
ADD	reg8,reg8	
ADD 注释	Reg8，reg8	8086
ADD	mem,reg16	8086,LOCK

ADD 注释	Mem，reg16	8086，LOCK
ADD	reg16,reg16	
ADD 注释	Reg16，reg16	8086
ADD	mem,reg32	386,LOCK
ADD 注释	Mem reg32	386，LOCK
ADD	reg32,reg32	
ADD 注释	Reg32，reg32	386
ADD	mem,reg64	X64,LOCK
ADD 注释	Mem，reg64	X64，锁定
ADD	reg64,reg64	X64
ADD 注释	Reg64，reg64	X64
ADD	reg8,mem	
ADD 注释	Reg8，mem	8086
ADD	reg8,reg8	
ADD 注释	Reg8，reg8	8086
ADD	reg16,mem	
ADD 注释	Reg16，mem	8086
ADD	reg16,reg16	
ADD 注释	Reg16，reg16	8086
ADD	reg32,mem	
ADD 注释	Reg32，mem	386
ADD	reg32,reg32	
ADD 注释	Reg32，reg32	386
ADD	reg64,mem	X64
ADD 注释	Reg64，mem	X64
ADD	reg64,reg64	X64
ADD 注释	Reg64，reg64	X64
ADD	rm16,imm8	8086,LOCK
ADD 注释	Rm16，imm8	8086，LOCK
ADD	rm32,imm8	386,LOCK
ADD 注释	Rm32，imm8	386，LOCK
ADD	rm64,imm8	X64,LOCK
ADD 注释	Rm64，imm8	X64，锁定
ADD	reg_al,imm	
ADD 注释	我叫 reg _ al	8086
ADD	reg_ax,sbyteword	8086,ND
ADD 注释	Reg _ ax，字节词	8086，ND
ADD	reg_ax,imm	
ADD 注释	Reg _ ax，imm	8086
ADD	reg_eax,sbytedword	386,ND
ADD 注释	Reg _ eax，bytedword	新界 386 号
ADD	reg_eax,imm	
ADD 注释	注意，注意	386
ADD	reg_rax,sbytedword	X64,ND
ADD 注释	Reg _ rax，字节词	X64，ND
ADD	reg_rax,imm	X64
ADD 注释	Reg _ rax，imm	X64
ADD	rm8,imm	8086,LOCK
ADD 注释	Rm8，imm	8086，LOCK
ADD	rm16,sbyteword	8086,LOCK,ND

ADD 注释	Rm16，字节词	新界洛克 8086 号
ADD	rm16,imm	8086,LOCK
ADD 注释	16 元	8086，LOCK
ADD	rm32,sbytedword	386,LOCK,ND
ADD 注释	Rm32，字节词	新界洛克 386 号
ADD	rm32,imm	386,LOCK
ADD 注释	Rm32，imm	386，LOCK
ADD	rm64,sbytedword	X64,LOCK,ND
ADD 注释	Rm64，字节词	X64，LOCK，ND

138

ADD	rm64,imm	X64,LOCK
ADD 注释	64 元	X64，锁定
ADD	mem,imm8	8086,LOCK
ADD 注释	Mem，imm8	8086，LOCK
ADD	mem,sbyteword16	8086,LOCK,ND
ADD 注释	Mem，byteword16	新界洛克 8086 号
ADD	mem,imm16	8086,LOCK
ADD 注释	我是 im16	8086，LOCK
ADD	mem,sbytedword32	386,LOCK,ND
ADD 注释	Mem，sbytedword32	新界洛克 386 号
ADD	mem,imm32	386,LOCK
ADD 注释	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
ADD	rm8,imm	8086，LOCK，ND，
ADD 注释	Rm8，imm	NOLONG
AND	mem,reg8	8086,LOCK
还有	Mem reg8	8086，LOCK
AND	reg8,reg8	
还有	Reg8，reg8	8086
AND	mem,reg16	8086,LOCK
还有	Mem，reg16	8086，LOCK
AND	reg16,reg16	
还有	Reg16，reg16	8086
AND	mem,reg32	386,LOCK
还有	Mem reg32	386，LOCK
AND	reg32,reg32	
还有	Reg32，reg32	386
AND	mem,reg64	X64,LOCK
还有	Mem，reg64	X64，锁定
AND	reg64,reg64	X64
还有	Reg64，reg64	X64
AND	reg8,mem	
还有	Reg8，mem	8086
AND	reg8,reg8	
还有	Reg8，reg8	8086
AND	reg16,mem	
还有	Reg16，mem	8086
AND	reg16,reg16	
还有	Reg16，reg16	8086
AND	reg32,mem	
还有	Reg32，mem	386
AND	reg32,reg32	
还有	Reg32，reg32	386
AND	reg64,mem	X64
还有	Reg64，mem	X64
AND	reg64,reg64	X64
还有	Reg64，reg64	X64
AND	rm16,imm8	8086,LOCK
还有	Rm16，imm8	8086，LOCK
AND	rm32,imm8	386,LOCK

还有	Rm32，imm8	386，LOCK
AND	rm64,imm8	X64,LOCK
还有	Rm64，imm8	X64，锁定
AND	reg_al,imm	
还有	我叫 reg _ al	8086
AND	reg_ax,sbyteword	8086,ND
还有	Reg _ ax，字节词	8086，ND
AND	reg_ax,imm	
还有	Reg _ ax，imm	8086
AND	reg_eax,sbytedword	386,ND
还有	Reg _ eax，bytedword	新界 386 号
AND	reg_eax,imm	
还有	注意，注意	386
AND	reg_rax,sbytedword	X64,ND
还有	Reg _ rax，字节词	X64，ND
AND	reg_rax,imm	X64
还有	Reg _ rax，imm	X64
AND	rm8,imm	8086,LOCK
还有	Rm8，imm	8086，LOCK
AND	rm16,sbyteword	8086,LOCK,ND
还有	Rm16，字节词	新界洛克 8086 号
AND	rm16,imm	8086,LOCK
还有	16 元	8086，LOCK
AND	rm32,sbytedword	386,LOCK,ND
还有	Rm32，字节词	新界洛克 386 号
AND	rm32,imm	386,LOCK
还有	Rm32，imm	386，LOCK
AND	rm64,sbytedword	X64,LOCK,ND
还有	Rm64，字节词	X64，LOCK，ND
AND	rm64,imm	X64,LOCK
还有	64 元	X64，锁定
AND	mem,imm8	8086,LOCK
还有	Mem，imm8	8086，LOCK
AND	mem,sbyteword16	8086,LOCK,ND
还有	Mem，byteword16	新界洛克 8086 号
AND	mem,imm16	8086,LOCK
还有	我是 im16	8086，LOCK
AND	mem,sbytedword32	386,LOCK,ND
还有	Mem，sbytedword32	新界洛克 386 号
AND	mem,imm32	386,LOCK
还有	我 32 岁	386，LOCK
AND	rm8,imm	8086,LOCK,ND,NOLONG
还有	Rm8，imm	8086，LOCK，ND，NOLONG
ARPL	mem,reg16	286,PROT,NOLONG
ARPL	Mem，reg16	286，PROT，NOLONG
ARPL	reg16,reg16	286,PROT,NOLONG
ARPL	Reg16，reg16	286，PROT，NOLONG
BB0_RESET		PENT,CYRIX,ND,OBSOLETE
Bb0 重置		废弃，CYRIX，ND，废弃

BB1_RESET		PENT,CYRIX,ND,OBSOLETE
Bb1 重置		废弃，CYRIX，ND，废弃
BOUND	reg16,mem	186,NOLONG
束缚	Reg16，mem	186，NOLONG
BOUND	reg32,mem	386,NOLONG
束缚	Reg32，mem	386，NOLONG
BSF	reg16,mem	
BSF	Reg16，mem	386
BSF	reg16,reg16	
BSF	Reg16，reg16	386

139

BSF	reg32,mem	
BSF	Reg32， mem	386
BSF	reg32,reg32	
BSF	Reg32， reg32	386
BSF	reg64,mem	X64
BSF	Reg64， mem	X64
BSF	reg64,reg64	X64
BSF	Reg64， reg64	X64
BSR	reg16,mem	
BSR	Reg16， mem	386
BSR	reg16,reg16	
BSR	Reg16， reg16	386
BSR	reg32,mem	
BSR	Reg32， mem	386
BSR	reg32,reg32	
BSR	Reg32， reg32	386
BSR	reg64,mem	X64
BSR	Reg64， mem	X64
BSR	reg64,reg64	X64
BSR	Reg64， reg64	X64
BSWAP	reg32	
交换计划	Reg32	486
BSWAP	reg64	X64
交换计划	Reg64	X64
BT	mem,reg16	
英国电信	Mem， reg16	386
BT	reg16,reg16	
英国电信	Reg16， reg16	386
BT	mem,reg32	
英国电信	Mem reg32	386
BT	reg32,reg32	
英国电信	Reg32， reg32	386
BT	mem,reg64	X64
英国电信	Mem， reg64	X64
BT	reg64,reg64	X64
英国电信	Reg64， reg64	X64
BT	rm16,imm	
英国电信	16 元	386
BT	rm32,imm	
英国电信	Rm32， imm	386
BT	rm64,imm	X64
英国电信	64 元	X64
BTC	mem,reg16	386,LOCK
BTC	Mem， reg16	386， LOCK
BTC	reg16,reg16	
BTC	Reg16， reg16	386
BTC	mem,reg32	386,LOCK
BTC	Mem reg32	386， LOCK
BTC	reg32,reg32	
BTC	Reg32， reg32	386

BTC	mem,reg64	X64,LOCK
BTC	Mem，reg64	X64，锁定
BTC	reg64,reg64	X64
BTC	Reg64，reg64	X64
BTC	rm16,imm	386,LOCK
BTC	16 元	386，LOCK
BTC	rm32,imm	386,LOCK
BTC	Rm32，imm	386，LOCK
BTC	rm64,imm	X64,LOCK
BTC	64 元	X64，锁定
BTR	mem,reg16	386,LOCK
顺便提一句	Mem，reg16	386，LOCK
BTR	reg16,reg16	
顺便提一句	Reg16，reg16	386
BTR	mem,reg32	386,LOCK
顺便提一句	Mem reg32	386，LOCK
BTR	reg32,reg32	
顺便提一句	Reg32，reg32	386
BTR	mem,reg64	X64,LOCK
顺便提一句	Mem，reg64	X64，锁定
BTR	reg64,reg64	X64
顺便提一句	Reg64，reg64	X64
BTR	rm16,imm	386,LOCK
顺便提一句	16 元	386，LOCK
BTR	rm32,imm	386,LOCK
顺便提一句	Rm32，imm	386，LOCK
BTR	rm64,imm	X64,LOCK
顺便提一句	64 元	X64，锁定
BTS	mem,reg16	386,LOCK
BTS	Mem，reg16	386，LOCK
BTS	reg16,reg16	
BTS	Reg16，reg16	386
BTS	mem,reg32	386,LOCK
BTS	Mem reg32	386，LOCK
BTS	reg32,reg32	
BTS	Reg32，reg32	386
BTS	mem,reg64	X64,LOCK
BTS	Mem，reg64	X64，锁定
BTS	reg64,reg64	X64
BTS	Reg64，reg64	X64
BTS	rm16,imm	386,LOCK
BTS	16 元	386，LOCK
BTS	rm32,imm	386,LOCK
BTS	Rm32，imm	386，LOCK
BTS	rm64,imm	X64,LOCK
BTS	64 元	X64，锁定
CALL	imm	8086,BND
打电话	是的	8086，BND
CALL	imm|near	8086,ND,BND
打电话	在附近	8086，ND，BND

		8086,ND,NOLONG
CALL	imm|far	NOLONG 新德里 8086
打电话	我很远	号
CALL	imm16	8086,NOLONG,BND
打电话	Im16	8086，NOLONG，BND
		8086,ND,NOLONG,BND
CALL	imm16|near	8086，ND，
打电话	Im16 | 近	NOLONG，BND
		8086,ND,NOLONG
CALL	imm16|far	NOLONG 新德里 8086
打电话	Imm16 | far	号

140

CALL	imm32	386,NOLONG,BND
打电话	Im32	386，NOLONG，BND
		386,ND,NOLONG,BND
CALL	imm32|near	386，ND，NOLONG，
打电话	Im32 | 近	BND
CALL	imm32|far	386,ND,NOLONG
打电话	Imm32 | far	386，ND，NOLONG
CALL	imm64	X64,BND
打电话	Im64	X64，BND
CALL	imm64|near	X64,ND,BND
打电话	Imm64 | 近	X64，ND，BND
CALL	imm:imm	8086,NOLONG
打电话	是的	8086，NOLONG
CALL	imm16:imm	8086,NOLONG
打电话	Imm16: imm	8086，NOLONG
CALL	imm:imm16	8086,NOLONG
打电话	Imm: imm16	8086，NOLONG
CALL	imm32:imm	386,NOLONG
打电话	Imm32: imm	386，NOLONG
CALL	imm:imm32	386,NOLONG
打电话	Imm: imm32	386，NOLONG
CALL	mem|far	8086,NOLONG
打电话	Mem | far	8086，NOLONG
CALL	mem|far	X64
打电话	Mem | far	X64
CALL	mem16|far	
打电话	Mem16 | far	8086
CALL	mem32|far	
打电话	Mem32 | far	386
CALL	mem64|far	X64
打电话	Mem64 | far	X64
CALL	mem|near	8086,ND,BND
打电话	在附近	8086，ND，BND
		8086,NOLONG,ND,BND
CALL	rm16|near	8086，NOLONG，
打电话	Rm16 | 近	ND，BND
		386,NOLONG,ND,BND
CALL	rm32|near	386，NOLONG，ND，
打电话	Rm32 | 近	BND
CALL	rm64|near	X64,ND,BND
打电话	Rm64 | 近	X64，ND，BND
CALL	mem	8086,BND
打电话	Mem	8086，BND
CALL	rm16	8086,NOLONG,BND
打电话	Rm16	8086，NOLONG，BND
CALL	rm32	386,NOLONG,BND
打电话	Rm32	386，NOLONG，BND
CALL	rm64	X64,BND
打电话	Rm64	X64，BND
CBW		8086

生化武器

CDQ		
CDQ		386
CDQE		X64
CDQE		X64
CLC		
CLC		8086
CLD		
CLD		8086
CLI		
CLI		8086
CLTS		286,PRIV
CLTS		286，PRIV
CMC		
CMC		8086
CMP	mem,reg8	
CMP	Mem reg8	8086
CMP	reg8,reg8	
CMP	Reg8，reg8	8086
CMP	mem,reg16	
CMP	Mem，reg16	8086
CMP	reg16,reg16	
CMP	Reg16，reg16	8086
CMP	mem,reg32	
CMP	Mem reg32	386
CMP	reg32,reg32	
CMP	Reg32，reg32	386
CMP	mem,reg64	X64
CMP	Mem，reg64	X64
CMP	reg64,reg64	X64
CMP	Reg64，reg64	X64
CMP	reg8,mem	
CMP	Reg8，mem	8086
CMP	reg8,reg8	
CMP	Reg8，reg8	8086
CMP	reg16,mem	
CMP	Reg16，mem	8086
CMP	reg16,reg16	
CMP	Reg16，reg16	8086
CMP	reg32,mem	
CMP	Reg32，mem	386
CMP	reg32,reg32	
CMP	Reg32，reg32	386
CMP	reg64,mem	X64
CMP	Reg64，mem	X64
CMP	reg64,reg64	X64
CMP	Reg64，reg64	X64
CMP	rm16,imm8	
CMP	Rm16，imm8	8086
CMP	rm32,imm8	
CMP	Rm32，imm8	386

CMP	rm64,imm8	X64
CMP	Rm64，imm8	X64
CMP	reg_al,imm	
CMP	我叫 reg _ al	8086
CMP	reg_ax,sbyteword	8086,ND
CMP	Reg _ ax，字节词	8086，ND
CMP	reg_ax,imm	
CMP	Reg _ ax，imm	8086
CMP	reg_eax,sbytedword	386,ND
CMP	Reg _ eax，bytedword	新界 386 号

141

CMP	reg_eax,imm	
CMP	注意，注意	386
CMP	reg_rax,sbytedword	X64,ND
CMP	Reg _ rax，字节词	X64，ND
CMP	reg_rax,imm	X64
CMP	Reg _ rax，imm	X64
CMP	rm8,imm	
CMP	Rm8，imm	8086
CMP	rm16,sbyteword	8086,ND
CMP	Rm16，字节词	8086，ND
CMP	rm16,imm	
CMP	16 元	8086
CMP	rm32,sbytedword	386,ND
CMP	Rm32，字节词	新界 386 号
CMP	rm32,imm	
CMP	Rm32，imm	386
CMP	rm64,sbytedword	X64,ND
CMP	Rm64，字节词	X64，ND
CMP	rm64,imm	X64
CMP	64 元	X64
CMP	mem,imm8	
CMP	Mem，imm8	8086
CMP	mem,sbyteword16	8086,ND
CMP	Mem，byteword16	8086，ND
CMP	mem,imm16	
CMP	我是 im16	8086
CMP	mem,sbytedword32	386,ND
CMP	Mem，sbytedword32	新界 386 号
CMP	mem,imm32	
CMP	我 32 岁	386
CMP	rm8,imm	8086,ND,NOLONG
CMP	Rm8，imm	NOLONG 新德里 8086 号
CMPSB		
公务员事务委员会		8086
CMPSD		
CMPSD		386
CMPSQ		
中国商品标准委员会		X64
		X64
CMPSW		
CMPSW		8086
CMPXCHG	mem,reg8	PENT,LOCK
CMPXCHG	Mem reg8	被压抑，锁定
CMPXCHG	reg8,reg8	PENT
CMPXCHG	Reg8，reg8	被压抑
CMPXCHG	mem,reg16	PENT,LOCK
CMPXCHG	Mem，reg16	被压抑，锁定
CMPXCHG	reg16,reg16	PENT
CMPXCHG	Reg16，reg16	被压抑

CMPXCHG	mem,reg32	PENT,LOCK
CMPXCHG	Mem reg32	被压抑，锁定
CMPXCHG	reg32,reg32	PENT
CMPXCHG	Reg32，reg32	被压抑
CMPXCHG	mem,reg64	X64,LOCK
CMPXCHG	Mem，reg64	X64，锁定
CMPXCHG	reg64,reg64	X64
CMPXCHG	Reg64，reg64	X64
CMPXCHG486	mem,reg8	486,UNDOC,ND,LOCK,OBSOLETE
CMPXCHG486	Mem reg8	486，UNDOC，ND，LOCK，OBSOLETE
CMPXCHG486	reg8,reg8	486,UNDOC,ND,OBSOLETE
CMPXCHG486	Reg8，reg8	486 UNDOC，ND，废弃
CMPXCHG486	mem,reg16	486,UNDOC,ND,LOCK,OBSOLETE
CMPXCHG486	Mem，reg16	486，UNDOC，ND，LOCK，OBSOLETE
CMPXCHG486	reg16,reg16	486,UNDOC,ND,OBSOLETE
CMPXCHG486	Reg16，reg16	486 UNDOC，ND，废弃
CMPXCHG486	mem,reg32	486,UNDOC,ND,LOCK,OBSOLETE
CMPXCHG486	Mem reg32	486，UNDOC，ND，LOCK，OBSOLETE
CMPXCHG486	reg32,reg32	486,UNDOC,ND,OBSOLETE
CMPXCHG486	Reg32，reg32	486 UNDOC，ND，废弃
CMPXCHG8B	mem	PENT,LOCK
CMPXCHG8B	Mem	被压抑，锁定
CMPXCHG16B	mem	X64,LOCK
CMPXCHG16B	Mem	X64，锁定
CPUID		PENT
CPUID		被压抑
CPU_READ		PENT,CYRIX
CPU _ read		被压抑的 CYRIX
CPU_WRITE		PENT,CYRIX
写		被压抑的 CYRIX
CQO		X64
CQO		X64
CWD		
CWD		8086
CWDE		
CWDE		386
DAA		8086,NOLONG
DAA		8086，NOLONG
DAS		8086,NOLONG
DAS		8086，NOLONG
DEC	reg16	8086,NOLONG
DEC	规例 16	8086，NOLONG
DEC	reg32	386,NOLONG
DEC	Reg32	386，NOLONG
DEC	rm8	8086,LOCK
DEC	8 令吉	8086，LOCK
DEC	rm16	
DEC	Rm16	8086,LOCK

		8086，LOCK
DEC	rm32	386,LOCK
DEC	Rm32	386，LOCK
DEC	rm64	X64,LOCK
DEC	Rm64	X64，锁定
DIV	rm8	
设计资料	8 令吉	8086
DIV	rm16	
设计资料	Rm16	8086
DIV	rm32	
设计资料	Rm32	386
DIV	rm64	X64
设计资料	Rm64	X64

142

DMINT		P6,CYRIX
DMINT		P6 CYRIX
EMMS		PENT,MMX
EMMS		废弃，MMX
ENTER	imm,imm	
回车	嗯，嗯	186
EQU	imm	
等式	是的	8086
EQU	imm:imm	
等式	是的	8086
F2XM1		8086,FPU
F2XM1		8086，FPU
FABS		8086,FPU
FABS		8086，FPU
FADD	mem32	8086,FPU
FADD	Mem32	8086，FPU
FADD	mem64	8086,FPU
FADD	Mem64	8086，FPU
FADD	fpureg|to	8086,FPU
FADD	Fpureg | to	8086，FPU
FADD	fpureg	8086,FPU
FADD	Fpureg	8086，FPU
FADD	fpureg,fpu0	8086,FPU
FADD	Fpureg，fpu0	8086，FPU
FADD	fpu0,fpureg	8086,FPU
FADD	Fpu0，fpureg	8086，FPU
		8086,FPU,ND
FADD		8086，FPU，
FADD		ND
FADDP	fpureg	8086,FPU
FADDP	Fpureg	8086，FPU
FADDP	fpureg,fpu0	8086,FPU
FADDP	Fpureg，fpu0	8086，FPU
		8086,FPU,ND
FADDP		8086，FPU，
FADDP		ND
FBLD	mem80	8086,FPU
联邦密探	Mem80	8086，FPU
FBLD	mem	8086,FPU
联邦密探	Mem	8086，FPU
FBSTP	mem80	8086,FPU
FBSTP	Mem80	8086，FPU
FBSTP	mem	8086,FPU
FBSTP	Mem	8086，FPU
FCHS		8086,FPU
FCHS		8086，FPU
FCLEX		8086,FPU
FCLEX FCLEX		8086，FPU
FCMOVB	fpureg	P6,FPU
FCMOVB	Fpureg	P6，FPU

FCMOVB	fpu0,fpureg	P6,FPU
FCMOVB	Fpu0，fpureg	P6，FPU
FCMOVB		P6,FPU,ND
FCMOVB		P6，FPU，ND
FCMOVBE	fpureg	P6,FPU
FCMOVBE	Fpureg	P6，FPU
FCMOVBE	fpu0,fpureg	P6,FPU
FCMOVBE	Fpu0，fpureg	P6，FPU
FCMOVBE		P6,FPU,ND
FCMOVBE		P6，FPU，ND
FCMOVE	fpureg	P6,FPU
FCMOVE	Fpureg	P6，FPU
FCMOVE	fpu0,fpureg	P6,FPU
FCMOVE	Fpu0，fpureg	P6，FPU
FCMOVE		P6,FPU,ND
FCMOVE		P6，FPU，ND
FCMOVNB	fpureg	P6,FPU
FCMOVNB	Fpureg	P6，FPU
FCMOVNB	fpu0,fpureg	P6,FPU
FCMOVNB	Fpu0，fpureg	P6，FPU
FCMOVNB		P6,FPU,ND
FCMOVNB		P6，FPU，ND
FCMOVNBE	fpureg	P6,FPU
FCMOVNBE	Fpureg	P6，FPU
FCMOVNBE	fpu0,fpureg	P6,FPU
FCMOVNBE	Fpu0，fpureg	P6，FPU
FCMOVNBE		P6,FPU,ND
FCMOVNBE		P6，FPU，ND
FCMOVNE		
FCMOVNE	fpureg	P6,FPU
FCMOVNE	Fpureg	P6，FPU
FCMOVNE		
FCMOVNE	fpu0,fpureg	P6,FPU
FCMOVNE	Fpu0，fpureg	P6，FPU
FCMOVNE		
FCMOVNE		P6,FPU,ND
FCMOVNE		P6，FPU，ND
FCMOVNU	fpureg	P6,FPU
FCMOVNU	Fpureg	P6，FPU
FCMOVNU	fpu0,fpureg	P6,FPU
FCMOVNU	Fpu0，fpureg	P6，FPU
FCMOVNU		P6,FPU,ND
FCMOVNU		P6，FPU，ND
FCMOVU	fpureg	P6,FPU
FCMOVU	Fpureg	P6，FPU
FCMOVU	fpu0,fpureg	P6,FPU
FCMOVU	Fpu0，fpureg	P6，FPU
FCMOVU		P6,FPU,ND
FCMOVU		P6，FPU，ND
	mem32	
FCOM	Mem32	8086,FPU

联邦通信委员		8086，FPU
会		
FCOM		
联邦通信委员	mem64	8086,FPU
会	Mem64	8086，FPU
FCOM		
联邦通信委员	fpureg	8086,FPU
会	Fpureg	8086，FPU
FCOM		
联邦通信委员	fpu0,fpureg	8086,FPU
会	Fpu0，fpureg	8086，FPU
FCOM		8086,FPU,ND
联邦通信委员		8086，FPU，
会		ND
FCOMI	fpureg	P6,FPU
FCOMI	Fpureg	P6，FPU
FCOMI	fpu0,fpureg	P6,FPU
FCOMI	Fpu0，fpureg	P6，FPU

143

FCOMI		P6,FPU,ND
FCOMI		P6，FPU，ND
FCOMIP	fpureg	P6,FPU
FCOMIP	Fpureg	P6，FPU
FCOMIP	fpu0,fpureg	P6,FPU
FCOMIP	Fpu0，fpureg	P6，FPU
FCOMIP		P6,FPU,ND
FCOMIP		P6，FPU，ND
FCOMP	mem32	8086,FPU
FCOMP	Mem32	8086，FPU
FCOMP	mem64	8086,FPU
FCOMP	Mem64	8086，FPU
FCOMP	fpureg	8086,FPU
FCOMP	Fpureg	8086，FPU
FCOMP	fpu0,fpureg	8086,FPU
FCOMP	Fpu0，fpureg	8086，FPU
FCOMP		8086,FPU,ND
FCOMP		8086，FPU，ND
FCOMPP		8086,FPU
FCOMPP		8086，FPU
FCOS		386,FPU
FCOS		386，FPU
FDECSTP		8086,FPU
FDECSTP		8086，FPU
FDISI		8086,FPU
FDISI		8086，FPU
FDIV	mem32	8086,FPU
FDIV	Mem32	8086，FPU
FDIV	mem64	8086,FPU
FDIV	Mem64	8086，FPU
FDIV	fpureg|to	8086,FPU
FDIV	Fpureg | to	8086，FPU
FDIV	fpureg	8086,FPU
FDIV	Fpureg	8086，FPU
FDIV	fpureg,fpu0	8086,FPU
FDIV	Fpureg，fpu0	8086，FPU
FDIV	fpu0,fpureg	8086,FPU
FDIV	Fpu0，fpureg	8086，FPU
FDIV		8086,FPU,ND
FDIV		8086，FPU，ND
FDIVP	fpureg	8086,FPU
FDIVP	Fpureg	8086，FPU
FDIVP	fpureg,fpu0	8086,FPU
FDIVP	Fpureg，fpu0	8086，FPU
FDIVP		8086,FPU,ND
FDIVP		8086，FPU，ND
FDIVR	mem32	8086,FPU
FDIVR	Mem32	8086，FPU
FDIVR	mem64	8086,FPU
FDIVR	Mem64	8086，FPU

FDIVR	fpureg|to	8086,FPU
FDIVR	Fpureg | to	8086，FPU
FDIVR	fpureg,fpu0	8086,FPU
FDIVR	Fpureg，fpu0	8086，FPU
FDIVR	fpureg	8086,FPU
FDIVR	Fpureg	8086，FPU
FDIVR	fpu0,fpureg	8086,FPU
FDIVR	Fpu0，fpureg	8086，FPU
FDIVR		8086,FPU,ND
FDIVR		8086，FPU，ND
FDIVRP	fpureg	8086,FPU
FDIVRP	Fpureg	8086，FPU
FDIVRP	fpureg,fpu0	8086,FPU
FDIVRP	Fpureg，fpu0	8086，FPU
FDIVRP		8086,FPU,ND
FDIVRP		8086，FPU，ND
FEMMS		PENT,3DNOW
FEMMS		废弃，3 DNOW
FENI		8086,FPU
芬尼		8086，FPU
FFREE	fpureg	8086,FPU
免费	Fpureg	8086，FPU
FFREE		8086,FPU
免费		8086，FPU
		286,FPU,UNDOC
FFREEP	fpureg	286，FPU，
FFREEP	Fpureg	UNDOC
		286,FPU,UNDOC
FFREEP		286，FPU，
FFREEP		UNDOC
FIADD	mem32	8086,FPU
FIADD FIADD	Mem32	8086，FPU
FIADD	mem16	8086,FPU
FIADD FIADD	Mem16	8086，FPU
FICOM	mem32	8086,FPU
FICOM	Mem32	8086，FPU
FICOM	mem16	8086,FPU
FICOM	Mem16	8086，FPU
FICOMP	mem32	8086,FPU
FICOMP	Mem32	8086，FPU
FICOMP	mem16	8086,FPU
FICOMP	Mem16	8086，FPU
FIDIV	mem32	8086,FPU
FIDIV FIDIV	Mem32	8086，FPU
FIDIV	mem16	8086,FPU
FIDIV FIDIV	Mem16	8086，FPU
FIDIVR	mem32	8086,FPU
FIDIVR FIDIVR	Mem32	8086，FPU
FIDIVR	mem16	8086,FPU
FIDIVR FIDIVR	Mem16	8086，FPU

FILD	mem32	8086,FPU
费尔德	Mem32	8086，FPU
FILD	mem16	8086,FPU
费尔德	Mem16	8086，FPU
FILD	mem64	8086,FPU
费尔德	Mem64	8086，FPU
FIMUL	mem32	8086,FPU
FIMUL	Mem32	8086，FPU
FIMUL	mem16	8086,FPU
FIMUL	Mem16	8086，FPU

144

FINCSTP		8086,FPU
FINCSTP		8086，FPU
FINIT		8086,FPU
终结		8086，FPU
FIST	mem32	8086,FPU
拳头	Mem32	8086，FPU
FIST	mem16	8086,FPU
拳头	Mem16	8086，FPU
FISTP	mem32	8086,FPU
拳头	Mem32	8086，FPU
FISTP	mem16	8086,FPU
拳头	Mem16	8086，FPU
FISTP	mem64	8086,FPU
拳头	Mem64	8086，FPU
FISTTP	mem16	PRESCOTT,FPU
拳击	Mem16	普雷斯科特，FPU
FISTTP	mem32	PRESCOTT,FPU
拳击	Mem32	普雷斯科特，FPU
FISTTP	mem64	PRESCOTT,FPU
拳击	Mem64	普雷斯科特，FPU
FISUB	mem32	8086,FPU
FISUB	Mem32	8086，FPU
FISUB	mem16	8086,FPU
FISUB	Mem16	8086，FPU
FISUBR	mem32	8086,FPU
FISUBR FISUBR	Mem32	8086，FPU
FISUBR	mem16	8086,FPU
FISUBR FISUBR	Mem16	8086，FPU
FLD	mem32	8086,FPU
FLD	Mem32	8086，FPU
FLD	mem64	8086,FPU
FLD	Mem64	8086，FPU
FLD	mem80	8086,FPU
FLD	Mem80	8086，FPU
FLD	fpureg	8086,FPU
FLD	Fpureg	8086，FPU
FLD		8086,FPU,ND
FLD		8086，FPU，ND
FLD1		8086,FPU
FLD1		8086，FPU
FLDCW	mem	8086,FPU,SW
FLDCW	Mem	8086，FPU，SW
FLDENV	mem	8086,FPU
弗兰登夫	Mem	8086，FPU
FLDL2E		8086,FPU

FLDL2E		8086，FPU
FLDL2T		8086,FPU
FLDL2T		8086，FPU
FLDLG2		8086,FPU
FLDLG2		8086，FPU
FLDLN2		8086,FPU
FLDLN2		8086，FPU
FLDPI		8086,FPU
FLDPI		8086，FPU
FLDZ		8086,FPU
FLDZ		8086，FPU
FMUL	mem32	8086,FPU
FMUL	Mem32	8086，FPU
FMUL	mem64	8086,FPU
FMUL	Mem64	8086，FPU
FMUL	fpureg|to	8086,FPU
FMUL	Fpureg | to	8086，FPU
FMUL	fpureg,fpu0	8086,FPU
FMUL	Fpureg，fpu0	8086，FPU
FMUL	fpureg	8086,FPU
FMUL	Fpureg	8086，FPU
FMUL	fpu0,fpureg	8086,FPU
FMUL	Fpu0，fpureg	8086，FPU
		8086,FPU,ND
FMUL		8086，FPU，
FMUL		ND
FMULP	fpureg	8086,FPU
FMULP	Fpureg	8086，FPU
FMULP	fpureg,fpu0	8086,FPU
FMULP	Fpureg，fpu0	8086，FPU
		8086,FPU,ND
FMULP		8086，FPU，
FMULP		ND
FNCLEX		8086,FPU
FNCLEX		8086，FPU
FNDISI		8086,FPU
FNDISI		8086，FPU
FNENI		8086,FPU
FNENI		8086，FPU
FNINIT		8086,FPU
FNINIT		8086，FPU
FNOP		8086,FPU
FNOP		8086，FPU
FNSAVE	mem	8086,FPU
FNSAVE	Mem	8086，FPU
		8086,FPU,SW
FNSTCW	mem	8086，FPU，
FNSTCW	Mem	SW
FNSTENV	mem	8086,FPU
FNSTENV	Mem	8086，FPU

		8086,FPU,SW
FNSTSW	mem	8086，FPU，
FNSTSW	Mem	SW
FNSTSW	reg_ax	286,FPU
FNSTSW	Reg _ ax	286，FPU
FPATAN		
FPATAN		8086,FPU
FPATAN		8086，FPU
FPREM		
FPREM 联邦调		8086,FPU
查局		8086，FPU
FPREM1		386,FPU
FPREM1		386，FPU
FPTAN		8086,FPU
FPTAN (FPTAN)		8086，FPU
FRNDINT		8086,FPU
Frndt 弗林丁特		8086，FPU
FRSTOR		
FRSTOR (冷冻	mem	8086,FPU
储存器)	Mem	8086，FPU

145

FSAVE	mem	8086,FPU
FSAVE (保存)	Mem	8086，FPU
FSCALE		8086,FPU
FSCALE		8086，FPU
FSETPM		286,FPU
FSETPM		286，FPU
FSIN		386,FPU
FSIN		386，FPU
FSINCOS		386,FPU
FSINCOS		386，FPU
FSQRT		8086,FPU
快速反应小组		8086，FPU
FST	mem32	8086,FPU
金融时报	Mem32	8086，FPU
FST	mem64	8086,FPU
金融时报	Mem64	8086，FPU
FST	fpureg	8086,FPU
金融时报	Fpureg	8086，FPU
		8086,FPU,ND
FST		8086，FPU，
金融时报		ND
		8086,FPU,SW
FSTCW	mem	8086，FPU，
FSTCW	Mem	SW
FSTENV	mem	8086,FPU
FSTENV	Mem	8086，FPU
FSTP	mem32	8086,FPU
FSTP	Mem32	8086，FPU
FSTP	mem64	8086,FPU
FSTP	Mem64	8086，FPU
FSTP	mem80	8086,FPU
FSTP	Mem80	8086，FPU
FSTP	fpureg	8086,FPU
FSTP	Fpureg	8086，FPU
		8086,FPU,ND
FSTP		8086，FPU，
FSTP		ND
		8086,FPU,SW
FSTSW	mem	8086，FPU，
FSTSW	Mem	SW
FSTSW	reg_ax	286,FPU
FSTSW	Reg _ ax	286，FPU
FSUB	mem32	8086,FPU
FSUB	Mem32	8086，FPU
FSUB	mem64	8086,FPU
FSUB	Mem64	8086，FPU
FSUB	fpureg|to	8086,FPU
FSUB	Fpureg | to	8086，FPU
FSUB	fpureg,fpu0	8086,FPU
FSUB	Fpureg，fpu0	8086，FPU

FSUB	fpureg	8086,FPU
FSUB	Fpureg	8086，FPU
FSUB	fpu0,fpureg	8086,FPU
FSUB	Fpu0，fpureg	8086，FPU
		8086,FPU,ND
FSUB		8086，FPU，
FSUB		ND
FSUBP	fpureg	8086,FPU
FSUBP	Fpureg	8086，FPU
FSUBP	fpureg,fpu0	8086,FPU
FSUBP	Fpureg，fpu0	8086，FPU
		8086,FPU,ND
FSUBP		8086，FPU，
FSUBP		ND
FSUBR	mem32	8086,FPU
FSUBR	Mem32	8086，FPU
FSUBR	mem64	8086,FPU
FSUBR	Mem64	8086，FPU
FSUBR	fpureg|to	8086,FPU
FSUBR	Fpureg | to	8086，FPU
FSUBR	fpureg,fpu0	8086,FPU
FSUBR	Fpureg，fpu0	8086，FPU
FSUBR	fpureg	8086,FPU
FSUBR	Fpureg	8086，FPU
FSUBR	fpu0,fpureg	8086,FPU
FSUBR	Fpu0，fpureg	8086，FPU
		8086,FPU,ND
FSUBR		8086，FPU，
FSUBR		ND
FSUBRP	fpureg	8086,FPU
FSUBRP	Fpureg	8086，FPU
FSUBRP	fpureg,fpu0	8086,FPU
FSUBRP	Fpureg，fpu0	8086，FPU
		8086,FPU,ND
FSUBRP		8086，FPU，
FSUBRP		ND
FTST		8086,FPU
FTST 金融时报		8086，FPU
FUCOM	fpureg	386,FPU
FUCOM	Fpureg	386，FPU
FUCOM	fpu0,fpureg	386,FPU
FUCOM	Fpu0，fpureg	386，FPU
		386,FPU,ND
FUCOM		386，FPU，
FUCOM		ND
FUCOMI	fpureg	P6,FPU
FUCOMI	Fpureg	P6，FPU
FUCOMI	fpu0,fpureg	P6,FPU
FUCOMI	Fpu0，fpureg	P6，FPU
FUCOMI		P6,FPU,ND

FUCOMI		P6，FPU，ND
FUCOMIP	fpureg	P6,FPU
FUCOMIP	Fpureg	P6，FPU
FUCOMIP	fpu0,fpureg	P6,FPU
FUCOMIP	Fpu0，fpureg	P6，FPU
FUCOMIP		P6,FPU,ND
FUCOMIP		P6，FPU，ND
FUCOMP	fpureg	386,FPU
FUCOMP 福康	Fpureg	386，FPU
FUCOMP	fpu0,fpureg	386,FPU
FUCOMP 福康	Fpu0，fpureg	386，FPU
		386,FPU,ND
FUCOMP		386，FPU，
FUCOMP 福康		ND
FUCOMPP		
FUCOMPP 富康		386,FPU
普公司		386，FPU
FXAM		8086,FPU
FXAM FXAM		8086，FPU

146

FXCH	fpureg	8086,FPU
FXCH	Fpureg	8086，FPU
FXCH	fpureg,fpu0	8086,FPU
FXCH	Fpureg，fpu0	8086，FPU
FXCH	fpu0,fpureg	8086,FPU
FXCH	Fpu0，fpureg	8086，FPU
FXCH		8086,FPU,ND
FXCH		8086，FPU，ND
FXTRACT		8086,FPU
FXTRACT		8086，FPU
FYL2X		8086,FPU
FYL2X		8086，FPU
FYL2XP1		8086,FPU
FYL2XP1		8086，FPU
HLT		8086,PRIV
高清晰度		8086，PRIV
		386,SW,UNDOC,ND,OBSOLETE
IBTS	mem,reg16	386，SW，UNDOC，ND，
IBTS	Mem，reg16	OBSOLETE
IBTS	reg16,reg16	386,UNDOC,ND,OBSOLETE
IBTS	Reg16，reg16	386，UNDOC，ND，OBSOLETE
		386,SD,UNDOC,ND,OBSOLETE
IBTS	mem,reg32	386，SD，UNDOC，ND，
IBTS	Mem reg32	OBSOLETE
IBTS	reg32,reg32	386,UNDOC,ND,OBSOLETE
IBTS	Reg32，reg32	386，UNDOC，ND，OBSOLETE
ICEBP		
ICEBP 英国石		386,ND
油公司		新界 386 号
IDIV	rm8	
IDIV	8 令吉	8086
IDIV	rm16	
IDIV	Rm16	8086
IDIV	rm32	
IDIV	Rm32	386
IDIV	rm64	X64
IDIV	Rm64	X64
IMUL	rm8	
IMUL	8 令吉	8086
IMUL	rm16	
IMUL	Rm16	8086
IMUL	rm32	
IMUL	Rm32	386
IMUL	rm64	X64
IMUL	Rm64	X64
IMUL	reg16,mem	
IMUL	Reg16，mem	386
IMUL	reg16,reg16	
IMUL	Reg16，reg16	386
IMUL	reg32,mem	386

IMUL	Reg32，mem	
IMUL	reg32,reg32	
IMUL	Reg32，reg32	386
IMUL	reg64,mem	X64
IMUL	Reg64，mem	X64
IMUL	reg64,reg64	X64
IMUL	Reg64，reg64	X64
IMUL	reg16,mem,imm8	
IMUL	Reg16，mem，imm8	186
IMUL	reg16,mem,sbyteword	186,ND
IMUL	Reg16，mem，byteword	186，ND
IMUL	reg16,mem,imm16	
IMUL	Reg16，mem，imm16	186
IMUL	reg16,mem,imm	186,ND
IMUL	Reg16，mem，imm	186，ND
IMUL	reg16,reg16,imm8	
IMUL	Reg16，reg16，imm8	186
IMUL	reg16,reg16,sbyteword	186,ND
IMUL	Reg16，reg16，字节词	186，ND
IMUL	reg16,reg16,imm16	
IMUL	Reg16，reg16，imm16	186
IMUL	reg16,reg16,imm	186,ND
IMUL	注意，注意，注意，注意	186，ND
IMUL	reg32,mem,imm8	
IMUL	Reg32，mem，imm8	386
IMUL	reg32,mem,sbytedword	386,ND
IMUL	Reg32，mem，bytedword	新界 386 号
IMUL	reg32,mem,imm32	
IMUL	Reg32，mem，imm32	386
IMUL	reg32,mem,imm	386,ND
IMUL	Reg32，mem，imm	新界 386 号
IMUL	reg32,reg32,imm8	
IMUL	Reg32 reg32 imm8	386
IMUL	reg32,reg32,sbytedword	386,ND
IMUL	Reg32，reg32，字节词	新界 386 号
IMUL	reg32,reg32,imm32	
IMUL	Reg32，reg32，imm32	386
IMUL	reg32,reg32,imm	386,ND
IMUL	Reg32，reg32，imm	新界 386 号
IMUL	reg64,mem,imm8	X64
IMUL	Reg64，mem，imm8	X64
IMUL	reg64,mem,sbytedword	X64,ND
IMUL	Reg64，mem，bytedword	X64，ND
IMUL	reg64,mem,imm32	X64
IMUL	Reg64，mem，imm32	X64
IMUL	reg64,mem,imm	X64,ND
IMUL	Reg64，mem，imm	X64，ND
IMUL	reg64,reg64,imm8	X64
IMUL	Reg64，reg64，imm8	X64
IMUL	reg64,reg64,sbytedword	X64,ND

IMUL	Reg64，reg64，bytedword	X64，ND
IMUL	reg64,reg64,imm32	X64
IMUL	Reg64 reg64 imm32	X64
IMUL	reg64,reg64,imm	X64,ND
IMUL	Reg64，reg64，imm	X64，ND
IMUL	reg16,imm8	
IMUL	Reg16，imm8	186
IMUL	reg16,sbyteword	186,ND
IMUL	Reg16，字节词	186，ND
IMUL	reg16,imm16	
IMUL	Reg16，imm16	186

147

IMUL	reg16,imm	186,ND
IMUL	第十六章	186，ND
IMUL	reg32,imm8	
IMUL	Reg32，imm8	386
IMUL	reg32,sbytedword	386,ND
IMUL	Reg32，bytedword	新界 386 号
IMUL	reg32,imm32	
IMUL	Reg32，imm32	386
IMUL	reg32,imm	386,ND
IMUL	三十二岁	新界 386 号
IMUL	reg64,imm8	X64
IMUL	Reg64，imm8	X64
IMUL	reg64,sbytedword	X64,ND
IMUL	Reg64，字节词	X64，ND
IMUL	reg64,imm32	X64
IMUL	Reg64，imm32	X64
IMUL	reg64,imm	X64,ND
IMUL	Reg64，imm	X64，ND
IN	reg_al,imm	
进来	我叫 reg _ al	8086
IN	reg_ax,imm	
进来	Reg _ ax，imm	8086
IN	reg_eax,imm	
进来	注意，注意	386
IN	reg_al,reg_dx	
进来	Reg _ al，reg _ dx	8086
IN	reg_ax,reg_dx	
进来	Reg _ ax，reg _ dx	8086
IN	reg_eax,reg_dx	
进来	Reg _ eax，reg _ dx	386
INC	reg16	8086,NOLONG
INC 公司	规例 16	8086，NOLONG
INC	reg32	386,NOLONG
INC 公司	Reg32	386，NOLONG
INC	rm8	8086,LOCK
INC 公司	8 令吉	8086，LOCK
INC	rm16	8086,LOCK
INC 公司	Rm16	8086，LOCK
INC	rm32	386,LOCK
INC 公司	Rm32	386，LOCK
INC	rm64	X64,LOCK
INC 公司	Rm64	X64，锁定
INSB		
INSB		186
INSD		
INSD		386
INSW		
新南威尔士州		186
INT	imm	
内景	是的	8086

INT01		386,ND
INT01		新界 386 号
INT1		
INT1		386
INT03		8086,ND
INT03		8086，ND
INT3		
INT3		8086
INTO		8086,NOLONG
进入		8086，NOLONG
INVD		
INVD 入境发展		486,PRIV
局		486 号，PRIV
		INVPCID,PRIV,NOLONG
INVPCID	reg32,mem128	INVPCID，PRIV，
INVPCID	Reg32，mem128	NOLONG
INVPCID	reg64,mem128	INVPCID,PRIV,LONG
INVPCID	Reg64，mem128	INVPCID，PRIV，LONG
INVLPG	mem	486,PRIV
INVLPG	Mem	486 号，PRIV
		X86_64,AMD,NOLONG
INVLPGA	reg_ax,reg_ecx	X86 _ 64，AMD，
INVLPGA	Reg _ ax，reg _ ecx	NOLONG
INVLPGA	reg_eax,reg_ecx	X86_64,AMD
INVLPGA	Reg _ eax，reg _ ecx	X86 _ 64，AMD
INVLPGA	reg_rax,reg_ecx	X64,AMD
INVLPGA	Reg _ rax，reg _ ecx	X64，AMD
INVLPGA		X86_64,AMD
INVLPGA		X86 _ 64，AMD
IRET		
IRET		8086
IRETD		
IRETD		386
IRETQ		X64
IRETQ		X64
IRETW		
IRETW		8086
JCXZ	imm	8086,NOLONG
JCXZ	是的	8086，NOLONG
JECXZ	imm	
JECXZ	是的	386
JRCXZ	imm	X64
JRCXZ	是的	X64
JMP	imm|short	
JMP	我很短	8086
JMP	imm	8086,ND
JMP	是的	8086，ND
JMP	imm	8086,BND
JMP	是的	8086，BND
JMP	imm|near	8086,ND,BND
JMP	在附近	8086，ND，BND

JMP	imm|far	8086,ND,NOLONG
JMP	我很远	NOLONG 新德里 8086 号
JMP	imm16	8086,NOLONG,BND
JMP	Im16	8086，NOLONG，BND
JMP	imm16|near	8086,ND,NOLONG,BND
JMP	Im16 | 近	8086，ND，NOLONG，BND
JMP	imm16|far	8086,ND,NOLONG
JMP	Imm16 | far	NOLONG 新德里 8086 号
JMP	imm32	386,NOLONG,BND
JMP	Im32	386，NOLONG，BND

148

JMP	imm32|near	386,ND,NOLONG,BND
JMP	Im32 | 近	386，ND，NOLONG，BND
JMP	imm32|far	386,ND,NOLONG
JMP	Imm32 | far	386，ND，NOLONG
JMP	imm64	X64,BND
JMP	Im64	X64，BND
JMP	imm64|near	X64,ND,BND
JMP	Imm64 | 近	X64，ND，BND
JMP	imm:imm	8086,NOLONG
JMP	是的	8086，NOLONG
JMP	imm16:imm	8086,NOLONG
JMP	Imm16: imm	8086，NOLONG
JMP	imm:imm16	8086,NOLONG
JMP	Imm: imm16	8086，NOLONG
JMP	imm32:imm	386,NOLONG
JMP	Imm32: imm	386，NOLONG
JMP	imm:imm32	386,NOLONG
JMP	Imm: imm32	386，NOLONG
JMP	mem|far	8086,NOLONG
JMP	Mem | far	8086，NOLONG
JMP	mem|far	X64
JMP	Mem | far	X64
JMP	mem16|far	
JMP	Mem16 | far	8086
JMP	mem32|far	
JMP	Mem32 | far	386
JMP	mem64|far	X64
JMP	Mem64 | far	X64
JMP	mem|near	8086,ND,BND
JMP	在附近	8086，ND，BND
JMP	rm16|near	8086,NOLONG,ND,BND
JMP	Rm16 | 近	8086，NOLONG，ND，BND
JMP	rm32|near	386,NOLONG,ND,BND
JMP	Rm32 | 近	386，NOLONG，ND，BND
JMP	rm64|near	X64,ND,BND
JMP	Rm64 | 近	X64，ND，BND
JMP	mem	8086,BND
JMP	Mem	8086，BND
JMP	rm16	8086,NOLONG,BND
JMP	Rm16	8086，NOLONG，BND
JMP	rm32	386,NOLONG,BND
JMP	Rm32	386，NOLONG，BND
JMP	rm64	X64,BND
JMP	Rm64	X64，BND
JMPE	imm	IA64
JMPE	是的	IA64
JMPE	imm16	IA64

JMPE	Im16	IA64
JMPE	imm32	IA64
JMPE	Im32	IA64
JMPE	rm16	IA64
JMPE	Rm16	IA64
JMPE	rm32	IA64
JMPE	Rm32	IA64
LAHF		
LAHF		8086
LAR	reg16,mem	286,PROT,SW
LAR	Reg16，mem	286，PROT，SW
LAR	reg16,reg16	286,PROT
LAR	Reg16，reg16	286，PROT
LAR	reg16,reg32	386,PROT
LAR	Reg16，reg32	386，PROT
LAR	reg16,reg64	X64,PROT,ND
LAR	Reg16，reg64	X64，PROT，ND
LAR	reg32,mem	386,PROT,SW
LAR	Reg32，mem	386，PROT，SW
LAR	reg32,reg16	386,PROT
LAR	Reg32，reg16	386，PROT
LAR	reg32,reg32	386,PROT
LAR	Reg32，reg32	386，PROT
LAR	reg32,reg64	X64,PROT,ND
LAR	Reg32，reg64	X64，PROT，ND
LAR	reg64,mem	X64,PROT,SW
LAR	Reg64，mem	X64，PROT，SW
LAR	reg64,reg16	X64,PROT
LAR	Reg64，reg16	X64，PROT
LAR	reg64,reg32	X64,PROT
LAR	Reg64，reg32	X64，PROT
LAR	reg64,reg64	X64,PROT
LAR	Reg64，reg64	X64，PROT
LDS	reg16,mem	8086,NOLONG
LDS	Reg16，mem	8086，NOLONG
LDS	reg32,mem	386,NOLONG
LDS	Reg32，mem	386，NOLONG
LEA	reg16,mem	
LEA	Reg16，mem	8086
LEA	reg32,mem	
LEA	Reg32，mem	386
LEA	reg64,mem	X64
LEA	Reg64，mem	X64
LEAVE		
离开		186
LES	reg16,mem	8086,NOLONG
LES	Reg16，mem	8086，NOLONG
LES	reg32,mem	386,NOLONG
LES	Reg32，mem	386，NOLONG
LFENCE		
LFENCE		X64,AMD

		X64，AMD
LFS	reg16,mem	
LFS	Reg16，mem	386
LFS	reg32,mem	
LFS	Reg32，mem	386
LFS	reg64,mem	X64
LFS	Reg64，mem	X64
LGDT	mem	286,PRIV
LGDT	Mem	286，PRIV
LGS	reg16,mem	
LGS	Reg16，mem	386

149

LGS	reg32,mem	
LGS	Reg32，mem	386
LGS	reg64,mem	X64
LGS	Reg64，mem	X64
LIDT	mem	286,PRIV
LIDT 利特	Mem	286，PRIV
LLDT	mem	286,PROT,PRIV
LLDT	Mem	286 号，PROT，PRIV
LLDT	mem16	286,PROT,PRIV
LLDT	Mem16	286 号，PROT，PRIV
LLDT	reg16	286,PROT,PRIV
LLDT	规例 16	286 号，PROT，PRIV
LMSW	mem	286,PRIV
LMSW	Mem	286，PRIV
LMSW	mem16	286,PRIV
LMSW	Mem16	286，PRIV
LMSW	reg16	286,PRIV
LMSW	规例 16	286，PRIV
LOADALL		386,UNDOC,ND,OBSOLETE
装载完毕		386，UNDOC，ND，OBSOLETE
LOADALL286		286,UNDOC,ND,OBSOLETE
LOADALL286		286，UNDOC，ND，OBSOLETE
LODSB		
LODSB		8086
LODSD		
LODSD		386
LODSQ		X64
LODSQ		X64
LODSW		
LODSW		8086
LOOP	imm	
循环	是的	8086
LOOP	imm,reg_cx	8086,NOLONG
循环	我是 reg _ cx	8086，NOLONG
LOOP	imm,reg_ecx	
循环	Imm，reg _ ecx	386
LOOP	imm,reg_rcx	X64
循环	Imm，reg _ rcx	X64
LOOPE	imm	
圈圈	是的	8086
LOOPE	imm,reg_cx	8086,NOLONG
圈圈	我是 reg _ cx	8086，NOLONG
LOOPE	imm,reg_ecx	
圈圈	Imm，reg _ ecx	386
LOOPE	imm,reg_rcx	X64
圈圈	Imm，reg _ rcx	X64
LOOPNE	imm	
LOOPNE	是的	8086

LOOPNE	imm,reg_cx	8086,NOLONG
LOOPNE	我是 reg _ cx	8086，NOLONG
LOOPNE	imm,reg_ecx	
LOOPNE	Imm，reg _ ecx	386
LOOPNE	imm,reg_rcx	X64
LOOPNE	Imm，reg _ rcx	X64
LOOPNZ	imm	
LOOPNZ	是的	8086
LOOPNZ	imm,reg_cx	8086,NOLONG
LOOPNZ	我是 reg _ cx	8086，NOLONG
LOOPNZ	imm,reg_ecx	
LOOPNZ	Imm，reg _ ecx	386
LOOPNZ	imm,reg_rcx	X64
LOOPNZ	Imm，reg _ rcx	X64
LOOPZ	imm	
LOOPZ	是的	8086
LOOPZ	imm,reg_cx	8086,NOLONG
LOOPZ	我是 reg _ cx	8086，NOLONG
LOOPZ	imm,reg_ecx	
LOOPZ	Imm，reg _ ecx	386
LOOPZ	imm,reg_rcx	X64
LOOPZ	Imm，reg _ rcx	X64
LSL	reg16,mem	286,PROT,SW
LSL	Reg16，mem	286，PROT，SW
LSL	reg16,reg16	286,PROT
LSL	Reg16，reg16	286，PROT
LSL	reg16,reg32	386,PROT
LSL	Reg16，reg32	386，PROT
LSL	reg16,reg64	X64,PROT,ND
LSL	Reg16，reg64	X64，PROT，ND
LSL	reg32,mem	386,PROT,SW
LSL	Reg32，mem	386，PROT，SW
LSL	reg32,reg16	386,PROT
LSL	Reg32，reg16	386，PROT
LSL	reg32,reg32	386,PROT
LSL	Reg32，reg32	386，PROT
LSL	reg32,reg64	X64,PROT,ND
LSL	Reg32，reg64	X64，PROT，ND
LSL	reg64,mem	X64,PROT,SW
LSL	Reg64，mem	X64，PROT，SW
LSL	reg64,reg16	X64,PROT
LSL	Reg64，reg16	X64，PROT
LSL	reg64,reg32	X64,PROT
LSL	Reg64，reg32	X64，PROT
LSL	reg64,reg64	X64,PROT
LSL	Reg64，reg64	X64，PROT
LSS	reg16,mem	
生活津贴	Reg16，mem	386
LSS	reg32,mem	
生活津贴	Reg32，mem	386

LSS	reg64,mem	X64
生活津贴	Reg64，mem	X64
LTR	mem	286,PROT,PRIV
轻型货柜码头	Mem	286 号，PROT，PRIV
LTR	mem16	286,PROT,PRIV
轻型货柜码头	Mem16	286 号，PROT，PRIV
LTR	reg16	286,PROT,PRIV
轻型货柜码头	规例 16	286 号，PROT，PRIV
MFENCE		X64,AMD
MFENCE		X64，AMD

150

MONITOR		PRESCOTT
监视器		普雷斯科特
		PRESCOTT,NOLONG,ND
MONITOR	reg_eax,reg_ecx,reg_edx	PRESCOTT，
监视器	Reg _ eax reg _ ecx reg _ edx	NOLONG，ND
MONITOR	reg_rax,reg_ecx,reg_edx	X64,ND
监视器	Reg _ rax，reg _ ecx，reg _ edx	X64，ND
MONITORX		AMD
监视器		AMD
MONITORX	reg_rax,reg_ecx,reg_edx	X64,AMD,ND
监视器	Reg _ rax，reg _ ecx，reg _ edx	X64，AMD，ND
MONITORX	reg_eax,reg_ecx,reg_edx	AMD,ND
监视器	Reg _ eax reg _ ecx reg _ edx	AMD，ND
MONITORX	reg_ax,reg_ecx,reg_edx	AMD,ND
监视器	Reg _ ax，reg _ ecx，reg _ edx	AMD，ND
MOV	mem,reg_sreg	8086,SW
MOV	Mem，reg _ sreg	西南 8086 号
MOV	reg16,reg_sreg	
MOV	Reg16，reg_sreg	8086
MOV	reg32,reg_sreg	
MOV	Reg32，reg_sreg	386
MOV	reg64,reg_sreg	X64,OPT,ND
MOV	Reg64，reg_sreg	X64，OPT，ND
MOV	rm64,reg_sreg	X64
MOV	Rm64，reg _ sreg	X64
MOV	reg_sreg,mem	8086,SW
MOV	Reg _ sreg mem	西南 8086 号
MOV	reg_sreg,reg16	8086,OPT,ND
MOV	Reg _ sreg，reg16	8086，OPT，ND
		386,OPT,ND
MOV	reg_sreg,reg32	新界巴勒斯坦被占领
MOV	Reg_sreg，reg32	土 386 号
MOV	reg_sreg,reg64	X64,OPT,ND
MOV	Reg_sreg，reg64	X64，OPT，ND
MOV	reg_sreg,reg16	
MOV	Reg _ sreg，reg16	8086
MOV	reg_sreg,reg32	
MOV	Reg_sreg，reg32	386
MOV	reg_sreg,rm64	X64
MOV	Reg _ sreg，rm64	X64
MOV	reg_al,mem_offs	
MOV	Reg _ al，mem _ off	8086
MOV	reg_ax,mem_offs	
MOV	注意，注意，注意	8086
MOV	reg_eax,mem_offs	
MOV	注意，注意，注意	386
MOV	reg_rax,mem_offs	X64
MOV	Reg _ rax，mem _ off	X64
MOV	mem_offs,reg_al	8086,NOHLE
MOV	记录，记录	8086，NOHLE

MOV	mem_offs,reg_ax	8086,NOHLE
MOV	记忆，记忆，记忆	8086，NOHLE
MOV	mem_offs,reg_eax	386,NOHLE
MOV	Mem _ off，reg _ eax	386，NOHLE
MOV	mem_offs,reg_rax	X64,NOHLE
MOV	Mem _ off reg _ rax	X64，NOHLE
MOV	reg32,reg_creg	386,PRIV,NOLONG
MOV	Reg32，reg_creg	386，PRIV，NOLONG
MOV	reg64,reg_creg	X64,PRIV
MOV	Reg64，reg_creg	X64，PRIV
MOV	reg_creg,reg32	386,PRIV,NOLONG
MOV	第 32 条	386，PRIV，NOLONG
MOV	reg_creg,reg64	X64,PRIV
MOV	Reg_creg，reg64	X64，PRIV
MOV	reg32,reg_dreg	386,PRIV,NOLONG
MOV	Reg32，reg_dreg	386，PRIV，NOLONG
MOV	reg64,reg_dreg	X64,PRIV
MOV	Reg64，reg_dreg	X64，PRIV
MOV	reg_dreg,reg32	386,PRIV,NOLONG
MOV	Reg _ dreg，reg32	386，PRIV，NOLONG
MOV	reg_dreg,reg64	X64,PRIV
MOV	Reg_dreg，reg64	X64，PRIV
MOV	reg32,reg_treg	386,NOLONG,ND
MOV	Reg32，reg_treg	386，NOLONG，ND
MOV	reg_treg,reg32	386,NOLONG,ND
MOV	Reg_treg，reg32	386，NOLONG，ND
MOV	mem,reg8	
MOV	Mem reg8	8086
MOV	reg8,reg8	
MOV	Reg8，reg8	8086
MOV	mem,reg16	
MOV	Mem，reg16	8086
MOV	reg16,reg16	
MOV	Reg16，reg16	8086
MOV	mem,reg32	
MOV	Mem reg32	386
MOV	reg32,reg32	
MOV	Reg32，reg32	386
MOV	mem,reg64	X64
MOV	Mem，reg64	X64
MOV	reg64,reg64	X64
MOV	Reg64，reg64	X64
MOV	reg8,mem	
MOV	Reg8，mem	8086
MOV	reg8,reg8	
MOV	Reg8，reg8	8086
MOV	reg16,mem	
MOV	Reg16，mem	8086
MOV	reg16,reg16	
MOV	Reg16，reg16	8086

MOV	reg32,mem	
MOV	Reg32， mem	386
MOV	reg32,reg32	
MOV	Reg32， reg32	386
MOV	reg64,mem	X64
MOV	Reg64， mem	X64
MOV	reg64,reg64	X64
MOV	Reg64， reg64	X64
MOV	reg8,imm	
MOV	Reg8， imm	8086

151

MOV	reg16,imm	
MOV	第十六章	8086
MOV	reg32,imm	
MOV	三十二岁	386
MOV	reg64,udword	X64,OPT,ND
MOV	Reg64，udword	X64，OPT，ND
MOV	reg64,sdword	X64,OPT,ND
MOV	Reg64，sdword	X64，OPT，ND
MOV	reg64,imm	X64
MOV	Reg64，imm	X64
MOV	rm8,imm	
MOV	Rm8，imm	8086
MOV	rm16,imm	
MOV	16 元	8086
MOV	rm32,imm	
MOV	Rm32，imm	386
MOV	rm64,imm	X64
MOV	64 元	X64
MOV	rm64,imm32	X64
MOV	Rm64，imm32	X64
MOV	mem,imm8	
MOV	Mem，imm8	8086
MOV	mem,imm16	
MOV	我是 im16	8086
MOV	mem,imm32	
MOV	我 32 岁	386
MOVD	mmxreg,rm32	PENT,MMX,SD
MOVD	Mmxreg，rm32	废气，MMX，SD
MOVD	rm32,mmxreg	PENT,MMX,SD
MOVD	Rm32，mmxreg	废气，MMX，SD
MOVD	mmxreg,rm64	X64,MMX,SX,ND
MOVD	Mmxreg，rm64	X64，MMX，SX，ND
MOVD	rm64,mmxreg	X64,MMX,SX,ND
MOVD	Rm64，mmxreg	X64，MMX，SX，ND
MOVQ	mmxreg,mmxrm	PENT,MMX
MOVQ	Mmxreg mmxrm	废弃，MMX
MOVQ	mmxrm,mmxreg	PENT,MMX
MOVQ	Mmxrm，mmxreg	废弃，MMX
MOVQ	mmxreg,rm64	X64,MMX
MOVQ	Mmxreg，rm64	X64，MMX
MOVQ	rm64,mmxreg	X64,MMX
MOVQ	Rm64，mmxreg	X64，MMX
MOVSB		
MOVSB		8086
MOVSD		
MOVSD		386

MOVSQ		X64
MOVSQ		X64
MOVSW		
MOVSW		8086
MOVSX	reg16,mem	
MOVSX	Reg16，mem	386
MOVSX	reg16,reg8	
MOVSX	Reg16，reg8	386
MOVSX	reg32,rm8	
MOVSX	Reg32，rm8	386
MOVSX	reg32,rm16	
MOVSX	Reg32，rm16	386
MOVSX	reg64,rm8	X64
MOVSX	Reg64，rm8	X64
MOVSX	reg64,rm16	X64
MOVSX	Reg64，rm16	X64
MOVSXD	reg64,rm32	X64
MOVSXD	Reg64，rm32	X64
MOVSX	reg64,rm32	X64,ND
MOVSX	Reg64，rm32	X64，ND
MOVZX	reg16,mem	
MOVZX	Reg16，mem	386
MOVZX	reg16,reg8	
MOVZX	Reg16，reg8	386
MOVZX	reg32,rm8	
MOVZX	Reg32，rm8	386
MOVZX	reg32,rm16	
MOVZX	Reg32，rm16	386
MOVZX	reg64,rm8	X64
MOVZX	Reg64，rm8	X64
MOVZX	reg64,rm16	X64
MOVZX	Reg64，rm16	X64
MUL	rm8	
MUL	8 令吉	8086
MUL	rm16	
MUL	Rm16	8086
MUL	rm32	
MUL	Rm32	386
MUL	rm64	X64
MUL	Rm64	X64
MWAIT		PRESCOTT
等等		普雷斯科特
		PRESCOTT,ND
MWAIT	reg_eax,reg_ecx	新德里州普雷斯
等等	Reg _ eax，reg _ ecx	科特
MWAITX		AMD
MWAITX		AMD
MWAITX	reg_eax,reg_ecx	AMD,ND
MWAITX	Reg _ eax，reg _ ecx	AMD，ND
NEG	rm8	8086,LOCK
NEG 否定	8 令吉	8086，LOCK

NEG	rm16	8086,LOCK
NEG 否定	Rm16	8086，LOCK
NEG	rm32	386,LOCK
NEG 否定	Rm32	386，LOCK
NEG	rm64	X64,LOCK
NEG 否定	Rm64	X64，锁定
NOP		
NOP		8086
NOP	rm16	P6
NOP	Rm16	P6
NOP	rm32	P6
NOP	Rm32	P6

152

NOP	rm64	X64
NOP	Rm64	X64
NOT	rm8	8086,LOCK
不是	8 令吉	8086，LOCK
NOT	rm16	8086,LOCK
不是	Rm16	8086，LOCK
NOT	rm32	386,LOCK
不是	Rm32	386，LOCK
NOT	rm64	X64,LOCK
不是	Rm64	X64，锁定
OR	mem,reg8	8086,LOCK
或者	Mem reg8	8086，LOCK
OR	reg8,reg8	
或者	Reg8，reg8	8086
OR	mem,reg16	8086,LOCK
或者	Mem，reg16	8086，LOCK
OR	reg16,reg16	
或者	Reg16，reg16	8086
OR	mem,reg32	386,LOCK
或者	Mem reg32	386，LOCK
OR	reg32,reg32	
或者	Reg32，reg32	386
OR	mem,reg64	X64,LOCK
或者	Mem，reg64	X64，锁定
OR	reg64,reg64	X64
或者	Reg64，reg64	X64
OR	reg8,mem	
或者	Reg8，mem	8086
OR	reg8,reg8	
或者	Reg8，reg8	8086
OR	reg16,mem	
或者	Reg16，mem	8086
OR	reg16,reg16	
或者	Reg16，reg16	8086
OR	reg32,mem	
或者	Reg32，mem	386
OR	reg32,reg32	
或者	Reg32，reg32	386
OR	reg64,mem	X64
或者	Reg64，mem	X64
OR	reg64,reg64	X64
或者	Reg64，reg64	X64
OR	rm16,imm8	8086,LOCK
或者	Rm16，imm8	8086，LOCK
OR	rm32,imm8	386,LOCK
或者	Rm32，imm8	386，LOCK
OR	rm64,imm8	X64,LOCK
或者	Rm64，imm8	X64，锁定
OR	reg_al,imm	
或者	我叫 reg _ al	8086

OR	reg_ax,sbyteword	8086,ND
或者	Reg _ ax，字节词	8086，ND
OR	reg_ax,imm	
或者	Reg _ ax，imm	8086
OR	reg_eax,sbytedword	386,ND
或者	Reg _ eax，bytedword	新界 386 号
OR	reg_eax,imm	
或者	注意，注意	386
OR	reg_rax,sbytedword	X64,ND
或者	Reg _ rax，字节词	X64，ND
OR	reg_rax,imm	X64
或者	Reg _ rax，imm	X64
OR	rm8,imm	8086,LOCK
或者	Rm8，imm	8086，LOCK
OR	rm16,sbyteword	8086,LOCK,ND
或者	Rm16，字节词	新界洛克 8086 号
OR	rm16,imm	8086,LOCK
或者	16 元	8086，LOCK
OR	rm32,sbytedword	386,LOCK,ND
或者	Rm32，字节词	新界洛克 386 号
OR	rm32,imm	386,LOCK
或者	Rm32，imm	386，LOCK
OR	rm64,sbytedword	X64,LOCK,ND
或者	Rm64，字节词	X64，LOCK，ND
OR	rm64,imm	X64,LOCK
或者	64 元	X64，锁定
OR	mem,imm8	8086,LOCK
或者	Mem，imm8	8086，LOCK
OR	mem,sbyteword16	8086,LOCK,ND
或者	Mem，byteword16	新界洛克 8086 号
OR	mem,imm16	8086,LOCK
或者	我是 im16	8086，LOCK
OR	mem,sbytedword32	386,LOCK,ND
或者	Mem，sbytedword32	新界洛克 386 号
OR	mem,imm32	386,LOCK
或者	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
OR	rm8,imm	8086，LOCK，ND，
或者	Rm8，imm	NOLONG
OUT	imm,reg_al	
出去	Imm，reg _ al	8086
OUT	imm,reg_ax	
出去	Imm，reg _ ax	8086
OUT	imm,reg_eax	
出去	Imm，reg _ eax	386
OUT	reg_dx,reg_al	
出去	注释 dx，注释 al	8086
OUT	reg_dx,reg_ax	
出去	Reg _ dx，reg _ ax	8086
OUT	reg_dx,reg_eax	386

出去	Reg _ dx，reg _ eax	
OUTSB		
出局		186
OUTSD		
出局		386
OUTSW		
外太空		186
PACKSSDW	mmxreg,mmxrm	PENT,MMX
PACKSSDW	Mmxreg mmxrm	废弃，MMX

153

PACKSSWB	mmxreg,mmxrm	PENT,MMX
PACKSSWB	Mmxreg mmxrm	废弃，MMX
PACKUSWB	mmxreg,mmxrm	PENT,MMX
PACKUSWB	Mmxreg mmxrm	废弃，MMX
PADDB	mmxreg,mmxrm	PENT,MMX
PADDB	Mmxreg mmxrm	废弃，MMX
PADDD	mmxreg,mmxrm	PENT,MMX
PADDD	Mmxreg mmxrm	废弃，MMX
PADDSB	mmxreg,mmxrm	PENT,MMX
PADDSB 帕德斯	Mmxreg mmxrm	废弃，MMX
		PENT,MMX,CYRIX
PADDSIW	mmxreg,mmxrm	PENT，MMX，
PADDSIW	Mmxreg mmxrm	CYRIX
PADDSW	mmxreg,mmxrm	PENT,MMX
PADDSW	Mmxreg mmxrm	废弃，MMX
PADDUSB	mmxreg,mmxrm	PENT,MMX
PADDUSB	Mmxreg mmxrm	废弃，MMX
PADDUSW	mmxreg,mmxrm	PENT,MMX
PADDUSW	Mmxreg mmxrm	废弃，MMX
PADDW	mmxreg,mmxrm	PENT,MMX
帕德沃	Mmxreg mmxrm	废弃，MMX
PAND	mmxreg,mmxrm	PENT,MMX
潘德	Mmxreg mmxrm	废弃，MMX
PANDN	mmxreg,mmxrm	PENT,MMX
潘登	Mmxreg mmxrm	废弃，MMX
PAUSE		
暂停		8086
		PENT,MMX,CYRIX
PAVEB	mmxreg,mmxrm	PENT，MMX，
PAVEB	Mmxreg mmxrm	CYRIX
PAVGUSB	mmxreg,mmxrm	PENT,3DNOW
PAVGUSB	Mmxreg mmxrm	废弃，3 DNOW
PCMPEQB	mmxreg,mmxrm	PENT,MMX
PCMPEQB	Mmxreg mmxrm	废弃，MMX
PCMPEQD	mmxreg,mmxrm	PENT,MMX
PCMPEQD	Mmxreg mmxrm	废弃，MMX
PCMPEQW	mmxreg,mmxrm	PENT,MMX
PCMPEQW	Mmxreg mmxrm	废弃，MMX
PCMPGTB	mmxreg,mmxrm	PENT,MMX
PCMPGTB	Mmxreg mmxrm	废弃，MMX
PCMPGTD	mmxreg,mmxrm	PENT,MMX
PCMPGTD	Mmxreg mmxrm	废弃，MMX
PCMPGTW	mmxreg,mmxrm	PENT,MMX
PCMPGTW	Mmxreg mmxrm	废弃，MMX
		PENT,MMX,CYRIX
PDISTIB	mmxreg,mem	PENT，MMX，
PDISTIB	Mmxreg mem	CYRIX
PF2ID	mmxreg,mmxrm	PENT,3DNOW
PF2ID	Mmxreg mmxrm	废弃，3 DNOW
PFACC	mmxreg,mmxrm	PENT,3DNOW

PFACC	Mmxreg mmxrm	废弃，3 DNOW
PFADD	mmxreg,mmxrm	PENT,3DNOW
PFADD PFADD	Mmxreg mmxrm	废弃，3 DNOW
PFCMPEQ	mmxreg,mmxrm	PENT,3DNOW
PFCMPEQ	Mmxreg mmxrm	废弃，3 DNOW
PFCMPGE	mmxreg,mmxrm	PENT,3DNOW
PFCMPGE	Mmxreg mmxrm	废弃，3 DNOW
PFCMPGT	mmxreg,mmxrm	PENT,3DNOW
PFCMPGT	Mmxreg mmxrm	废弃，3 DNOW
PFMAX	mmxreg,mmxrm	PENT,3DNOW
PFMAX PFMAX	Mmxreg mmxrm	废弃，3 DNOW
PFMIN	mmxreg,mmxrm	PENT,3DNOW
PFMIN	Mmxreg mmxrm	废弃，3 DNOW
PFMUL	mmxreg,mmxrm	PENT,3DNOW
PFMUL	Mmxreg mmxrm	废弃，3 DNOW
PFRCP	mmxreg,mmxrm	PENT,3DNOW
PFRCP	Mmxreg mmxrm	废弃，3 DNOW
PFRCPIT1	mmxreg,mmxrm	PENT,3DNOW
PFRCPIT1	Mmxreg mmxrm	废弃，3 DNOW
PFRCPIT2	mmxreg,mmxrm	PENT,3DNOW
PFRCPIT2	Mmxreg mmxrm	废弃，3 DNOW
PFRSQIT1	mmxreg,mmxrm	PENT,3DNOW
PFRSQIT1	Mmxreg mmxrm	废弃，3 DNOW
PFRSQRT	mmxreg,mmxrm	PENT,3DNOW
PFRSQRT	Mmxreg mmxrm	废弃，3 DNOW
PFSUB	mmxreg,mmxrm	PENT,3DNOW
PFSUB	Mmxreg mmxrm	废弃，3 DNOW
PFSUBR	mmxreg,mmxrm	PENT,3DNOW
PFSUBR	Mmxreg mmxrm	废弃，3 DNOW
PI2FD	mmxreg,mmxrm	PENT,3DNOW
PI2FD	Mmxreg mmxrm	废弃，3 DNOW
		PENT,MMX,CYRIX
PMACHRIW	mmxreg,mem	PENT，MMX，
PMACHRIW	Mmxreg mem	CYRIX
PMADDWD	mmxreg,mmxrm	PENT,MMX
PMADDWD	Mmxreg mmxrm	废弃，MMX
		PENT,MMX,CYRIX
PMAGW	mmxreg,mmxrm	PENT，MMX，
PMAGW	Mmxreg mmxrm	CYRIX
		PENT,MMX,CYRIX
PMULHRIW	mmxreg,mmxrm	PENT，MMX，
PMULHRIW	Mmxreg mmxrm	CYRIX
PMULHRWA	mmxreg,mmxrm	PENT,3DNOW
PMULHRWA	Mmxreg mmxrm	废弃，3 DNOW
		PENT,MMX,CYRIX
PMULHRWC	mmxreg,mmxrm	PENT，MMX，
PMULHRWC	Mmxreg mmxrm	CYRIX
PMULHW	mmxreg,mmxrm	PENT,MMX
PMULHW	Mmxreg mmxrm	废弃，MMX
PMULLW	mmxreg,mmxrm	PENT,MMX

PMULLW	Mmxreg mmxrm	废弃，MMX
		PENT,MMX,CYRIX
PMVGEZB	mmxreg,mem	PENT，MMX，
PMVGEZB	Mmxreg mem	CYRIX
		PENT,MMX,CYRIX
PMVLZB	mmxreg,mem	PENT，MMX，
PMVLZB	Mmxreg mem	CYRIX
		PENT,MMX,CYRIX
PMVNZB	mmxreg,mem	PENT，MMX，
PMVNZB	Mmxreg mem	CYRIX
		PENT,MMX,CYRIX
PMVZB	mmxreg,mem	PENT，MMX，
PMVZB	Mmxreg mem	CYRIX
POP	reg16	
POP 流行音乐	规例 16	8086
POP	reg32	386,NOLONG
POP 流行音乐	Reg32	386，NOLONG
POP	reg64	X64
POP 流行音乐	Reg64	X64

154

POP	rm16	
POP 流行音乐	Rm16	8086
POP	rm32	386,NOLONG
POP 流行音乐	Rm32	386，NOLONG
POP	rm64	X64
POP 流行音乐	Rm64	X64
POP	reg_es	8086,NOLONG
POP 流行音乐	规章制度	8086，NOLONG
POP	reg_cs	8086,UNDOC,ND,OBSOLETE
POP 流行音乐	注意事项	8086，UNDOC，ND，废弃
POP	reg_ss	8086,NOLONG
POP 流行音乐	注册表格	8086，NOLONG
POP	reg_ds	8086,NOLONG
POP 流行音乐	注意事项	8086，NOLONG
POP	reg_fs	
POP 流行音乐	Reg _ fs	386
POP	reg_gs	
POP 流行音乐	Reg _ gs	386
POPA		186,NOLONG
POPA		186，NOLONG
POPAD		386,NOLONG
POPAD (波帕)		386，NOLONG
POPAW		186,NOLONG
外婆		186，NOLONG
POPF		
波普夫		8086
POPFD		386,NOLONG
POPFD		386，NOLONG
POPFQ		X64
POPFQ		X64
POPFW		
POPFW		8086
POR	mmxreg,mmxrm	PENT,MMX
POR	Mmxreg mmxrm	废弃，MMX
PREFETCH	mem	PENT,3DNOW
预取	Mem	废弃，3 DNOW
PREFETCHW		
PREFETCHW 预	mem	PENT,3DNOW
备	Mem	废弃，3 DNOW
PSLLD	mmxreg,mmxrm	PENT,MMX
PSLLD	Mmxreg mmxrm	废弃，MMX
PSLLD	mmxreg,imm	PENT,MMX
PSLLD	我不知道	废弃，MMX
PSLLQ	mmxreg,mmxrm	PENT,MMX
PSLLQ	Mmxreg mmxrm	废弃，MMX
PSLLQ	mmxreg,imm	PENT,MMX
PSLLQ	我不知道	废弃，MMX
PSLLW	mmxreg,mmxrm	PENT,MMX
PSLLW	Mmxreg mmxrm	废弃，MMX
PSLLW	mmxreg,imm	PENT,MMX

PSLLW	我不知道	废弃，MMX
PSRAD	mmxreg,mmxrm	PENT,MMX
PSRAD	Mmxreg mmxrm	废弃，MMX
PSRAD	mmxreg,imm	PENT,MMX
PSRAD	我不知道	废弃，MMX
PSRAW	mmxreg,mmxrm	PENT,MMX
PSRAW	Mmxreg mmxrm	废弃，MMX
PSRAW	mmxreg,imm	PENT,MMX
PSRAW	我不知道	废弃，MMX
PSRLD	mmxreg,mmxrm	PENT,MMX
PSRLD	Mmxreg mmxrm	废弃，MMX
PSRLD	mmxreg,imm	PENT,MMX
PSRLD	我不知道	废弃，MMX
PSRLQ	mmxreg,mmxrm	PENT,MMX
PSRLQ	Mmxreg mmxrm	废弃，MMX
PSRLQ	mmxreg,imm	PENT,MMX
PSRLQ	我不知道	废弃，MMX
PSRLW	mmxreg,mmxrm	PENT,MMX
PSRLW	Mmxreg mmxrm	废弃，MMX
PSRLW	mmxreg,imm	PENT,MMX
PSRLW	我不知道	废弃，MMX
PSUBB	mmxreg,mmxrm	PENT,MMX
PSUBB	Mmxreg mmxrm	废弃，MMX
PSUBD	mmxreg,mmxrm	PENT,MMX
PSUBD	Mmxreg mmxrm	废弃，MMX
PSUBSB	mmxreg,mmxrm	PENT,MMX
PSUBSB	Mmxreg mmxrm	废弃，MMX
PSUBSIW	mmxreg,mmxrm	PENT,MMX,CYRIX
Psubw	Mmxreg mmxrm	PENT，MMX，CYRIX
PSUBSW	mmxreg,mmxrm	PENT,MMX
PSUBSW	Mmxreg mmxrm	废弃，MMX
PSUBUSB	mmxreg,mmxrm	PENT,MMX
PSUBUSB	Mmxreg mmxrm	废弃，MMX
PSUBUSW	mmxreg,mmxrm	PENT,MMX
PSUBUSW	Mmxreg mmxrm	废弃，MMX
PSUBW	mmxreg,mmxrm	PENT,MMX
PSUBW	Mmxreg mmxrm	废弃，MMX
PUNPCKHBW	mmxreg,mmxrm	PENT,MMX
PUNPCKHBW	Mmxreg mmxrm	废弃，MMX
PUNPCKHDQ	mmxreg,mmxrm	PENT,MMX
PUNPCKHDQ	Mmxreg mmxrm	废弃，MMX
PUNPCKHWD	mmxreg,mmxrm	PENT,MMX
庞普克人	Mmxreg mmxrm	废弃，MMX
PUNPCKLBW	mmxreg,mmxrm	PENT,MMX
彭彭	Mmxreg mmxrm	废弃，MMX
PUNPCKLDQ	mmxreg,mmxrm	PENT,MMX
PUNPCKLDQ	Mmxreg mmxrm	废弃，MMX
PUNPCKLWD	mmxreg,mmxrm	PENT,MMX
PUNPCKLWD	Mmxreg mmxrm	废弃，MMX
PUSH	reg16	8086

用力	规例 16	
PUSH	reg32	386,NOLONG
用力	Reg32	386，NOLONG
PUSH	reg64	X64
用力	Reg64	X64
PUSH	rm16	
用力	Rm16	8086
PUSH	rm32	386,NOLONG
用力	Rm32	386，NOLONG

155

PUSH	rm64	X64
用力	Rm64	X64
PUSH	reg_es	8086,NOLONG
用力	规章制度	8086，NOLONG
PUSH	reg_cs	8086,NOLONG
用力	注意事项	8086，NOLONG
PUSH	reg_ss	8086,NOLONG
用力	注册表格	8086，NOLONG
PUSH	reg_ds	8086,NOLONG
用力	注意事项	8086，NOLONG
PUSH	reg_fs	
用力	Reg _ fs	386
PUSH	reg_gs	
用力	Reg _ gs	386
PUSH	imm8	
用力	Im8	186
PUSH	sbyteword16	186,AR0,SIZE,ND
用力	字节 16	186，AR0，SIZE，ND
PUSH	imm16	186,AR0,SIZE
用力	Im16	186，AR0，尺寸
		386,NOLONG,AR0,SIZE,ND
PUSH	sbytedword32	386，NOLONG，AR0，
用力	Sbytedword32	SIZE，ND
PUSH	imm32	386,NOLONG,AR0,SIZE
用力	Im32	386，NOLONG，AR0，尺寸
PUSH	sbytedword32	386,NOLONG,SD,ND
用力	Sbytedword32	386，NOLONG，SD，ND
PUSH	imm32	386,NOLONG,SD
用力	Im32	386，NOLONG，SD
PUSH	sbytedword64	X64,AR0,SIZE,ND
用力	字节词 64	X64，AR0，SIZE，ND
PUSH	imm64	X64,AR0,SIZE
用力	Im64	X64，AR0，SIZE
PUSH	sbytedword32	X64,AR0,SIZE,ND
用力	Sbytedword32	X64，AR0，SIZE，ND
PUSH	imm32	X64,AR0,SIZE
用力	Im32	X64，AR0，SIZE
PUSHA		186,NOLONG
普夏		186，NOLONG
PUSHAD		386,NOLONG
普沙德		386，NOLONG
PUSHAW		
普肖		186,NOLONG
(PUSHAW)		186，NOLONG
PUSHF		
普希夫		8086
PUSHFD		386,NOLONG
PUSHFD		386，NOLONG
PUSHFQ		X64
PUSHFQ		X64

PUSHFW		
PUSHFW		8086
PXOR	mmxreg,mmxrm	PENT,MMX
PXOR	Mmxreg mmxrm	废弃，MMX
RCL	rm8,unity	
RCL	8 林吉特，团结	8086
RCL	rm8,reg_cl	
RCL	Rm8，reg _ cl	8086
RCL	rm8,imm8	
RCL	Rm8，imm8	186
RCL	rm16,unity	
RCL	Rm16，团结	8086
RCL	rm16,reg_cl	
RCL	Rm16，reg _ cl	8086
RCL	rm16,imm8	
RCL	Rm16，imm8	186
RCL	rm32,unity	
RCL	Rm32，团结	386
RCL	rm32,reg_cl	
RCL	Rm32，reg _ cl	386
RCL	rm32,imm8	
RCL	Rm32，imm8	386
RCL	rm64,unity	X64
RCL	Rm64，团结	X64
RCL	rm64,reg_cl	X64
RCL	Rm64，reg _ cl	X64
RCL	rm64,imm8	X64
RCL	Rm64，imm8	X64
RCR	rm8,unity	
RCR	8 林吉特，团结	8086
RCR	rm8,reg_cl	
RCR	Rm8，reg _ cl	8086
RCR	rm8,imm8	
RCR	Rm8，imm8	186
RCR	rm16,unity	
RCR	Rm16，团结	8086
RCR	rm16,reg_cl	
RCR	Rm16，reg _ cl	8086
RCR	rm16,imm8	
RCR	Rm16，imm8	186
RCR	rm32,unity	
RCR	Rm32，团结	386
RCR	rm32,reg_cl	
RCR	Rm32，reg _ cl	386
RCR	rm32,imm8	
RCR	Rm32，imm8	386
RCR	rm64,unity	X64
RCR	Rm64，团结	X64
RCR	rm64,reg_cl	X64
RCR	Rm64，reg _ cl	X64

RCR	rm64,imm8	X64
RCR	Rm64，imm8	X64
RDSHR	rm32	P6,CYRIX,SMM
RDSHR	Rm32	P6，CYRIX，SMM
RDMSR		PENT,PRIV
Rdmmsr		废弃的，私立学校
RDPMC		P6
RDPMC		P6
RDTSC		PENT
RDTSC		被压抑

156

RDTSCP		X86_64
RDTSCP		X86 _ 64
RET		8086,BND
RET		8086，BND
RET	imm	8086,SW,BND
RET	是的	8086，SW，BND
RETF		
反恐特遣队		8086
RETF	imm	8086,SW
反恐特遣队	是的	西南 8086 号
RETN		8086,BND
RETN		8086，BND
RETN	imm	8086,SW,BND
RETN	是的	8086，SW，BND
RETW		8086,BND
RETW		8086，BND
RETW	imm	8086,SW,BND
RETW	是的	8086，SW，BND
RETFW		
RETFW		8086
RETFW	imm	8086,SW
RETFW	是的	西南 8086 号
RETNW		8086,BND
RETNW 报道		8086，BND
RETNW	imm	8086,SW,BND
RETNW 报道	是的	8086，SW，BND
RETD		8086,BND,NOLONG
RETD		8086，BND，NOLONG
		8086,SW,BND,NOLONG
RETD	imm	8086，SW，BND，
RETD	是的	NOLONG
RETFD		
RETFD		8086
RETFD	imm	8086,SW
RETFD	是的	西南 8086 号
RETND		8086,BND,NOLONG
返回页面		8086，BND，NOLONG
		8086,SW,BND,NOLONG
RETND	imm	8086，SW，BND，
返回页面	是的	NOLONG
RETQ		X64,BND
RETQ		X64，BND
RETQ	imm	X64,SW,BND
RETQ	是的	X64，SW，BND
RETFQ		X64
RETFQ		X64
RETFQ	imm	X64,SW
RETFQ	是的	X64，SW
RETNQ		X64,BND
RETNQ		X64，BND

RETNQ	imm	X64,SW,BND
RETNQ	是的	X64，SW，BND
ROL	rm8,unity	
ROL	8 林吉特，团结	8086
ROL	rm8,reg_cl	
ROL	Rm8，reg _ cl	8086
ROL	rm8,imm8	
ROL	Rm8，imm8	186
ROL	rm16,unity	
ROL	Rm16，团结	8086
ROL	rm16,reg_cl	
ROL	Rm16，reg _ cl	8086
ROL	rm16,imm8	
ROL	Rm16，imm8	186
ROL	rm32,unity	
ROL	Rm32，团结	386
ROL	rm32,reg_cl	
ROL	Rm32，reg _ cl	386
ROL	rm32,imm8	
ROL	Rm32，imm8	386
ROL	rm64,unity	X64
ROL	Rm64，团结	X64
ROL	rm64,reg_cl	X64
ROL	Rm64，reg _ cl	X64
ROL	rm64,imm8	X64
ROL	Rm64，imm8	X64
ROR	rm8,unity	
ROR	8 林吉特，团结	8086
ROR	rm8,reg_cl	
ROR	Rm8，reg _ cl	8086
ROR	rm8,imm8	
ROR	Rm8，imm8	186
ROR	rm16,unity	
ROR	Rm16，团结	8086
ROR	rm16,reg_cl	
ROR	Rm16，reg _ cl	8086
ROR	rm16,imm8	
ROR	Rm16，imm8	186
ROR	rm32,unity	
ROR	Rm32，团结	386
ROR	rm32,reg_cl	
ROR	Rm32，reg _ cl	386
ROR	rm32,imm8	
ROR	Rm32，imm8	386
ROR	rm64,unity	X64
ROR	Rm64，团结	X64
ROR	rm64,reg_cl	X64
ROR	Rm64，reg _ cl	X64
ROR	rm64,imm8	X64
ROR	Rm64，imm8	X64

RDM		P6,CYRIX,ND
RDM		P6，CYRIX，ND
RSDC	reg_sreg,mem80	486,CYRIX,SMM
RSDC	条例草案，mem80	486，CYRIX，SMM
RSLDT	mem80	486,CYRIX,SMM
RSLDT	Mem80	486，CYRIX，SMM
RSM		PENT,SMM
RSM		被压抑的 SMM
RSTS	mem80	486,CYRIX,SMM
RSTS	Mem80	486，CYRIX，SMM

157

SAHF		
SAHF		8086
SAL	rm8,unity	8086,ND
萨尔	8 林吉特，团结	8086，ND
SAL	rm8,reg_cl	8086,ND
萨尔	Rm8，reg _ cl	8086，ND
SAL	rm8,imm8	186,ND
萨尔	Rm8，imm8	186，ND
SAL	rm16,unity	8086,ND
萨尔	Rm16，团结	8086，ND
SAL	rm16,reg_cl	8086,ND
萨尔	Rm16，reg _ cl	8086，ND
SAL	rm16,imm8	186,ND
萨尔	Rm16，imm8	186，ND
SAL	rm32,unity	386,ND
萨尔	Rm32，团结	新界 386 号
SAL	rm32,reg_cl	386,ND
萨尔	Rm32，reg _ cl	新界 386 号
SAL	rm32,imm8	386,ND
萨尔	Rm32，imm8	新界 386 号
SAL	rm64,unity	X64,ND
萨尔	Rm64，团结	X64，ND
SAL	rm64,reg_cl	X64,ND
萨尔	Rm64，reg _ cl	X64，ND
SAL	rm64,imm8	X64,ND
萨尔	Rm64，imm8	X64，ND
SALC		8086,UNDOC
SALC		8086 UNDOC
SAR	rm8,unity	
特别行政区	8 林吉特，团结	8086
SAR	rm8,reg_cl	
特别行政区	Rm8，reg _ cl	8086
SAR	rm8,imm8	
特别行政区	Rm8，imm8	186
SAR	rm16,unity	
特别行政区	Rm16，团结	8086
SAR	rm16,reg_cl	
特别行政区	Rm16，reg _ cl	8086
SAR	rm16,imm8	
特别行政区	Rm16，imm8	186
SAR	rm32,unity	
特别行政区	Rm32，团结	386
SAR	rm32,reg_cl	
特别行政区	Rm32，reg _ cl	386
SAR	rm32,imm8	
特别行政区	Rm32，imm8	386
SAR	rm64,unity	X64
特别行政区	Rm64，团结	X64
SAR	rm64,reg_cl	X64
特别行政区	Rm64，reg _ cl	X64

SAR	rm64,imm8	X64
特别行政区	Rm64，imm8	X64
SBB	mem,reg8	8086,LOCK
SBB	Mem reg8	8086，LOCK
SBB	reg8,reg8	
SBB	Reg8，reg8	8086
SBB	mem,reg16	8086,LOCK
SBB	Mem，reg16	8086，LOCK
SBB	reg16,reg16	
SBB	Reg16，reg16	8086
SBB	mem,reg32	386,LOCK
SBB	Mem reg32	386，LOCK
SBB	reg32,reg32	
SBB	Reg32，reg32	386
SBB	mem,reg64	X64,LOCK
SBB	Mem，reg64	X64，锁定
SBB	reg64,reg64	X64
SBB	Reg64，reg64	X64
SBB	reg8,mem	
SBB	Reg8，mem	8086
SBB	reg8,reg8	
SBB	Reg8，reg8	8086
SBB	reg16,mem	
SBB	Reg16，mem	8086
SBB	reg16,reg16	
SBB	Reg16，reg16	8086
SBB	reg32,mem	
SBB	Reg32，mem	386
SBB	reg32,reg32	
SBB	Reg32，reg32	386
SBB	reg64,mem	X64
SBB	Reg64，mem	X64
SBB	reg64,reg64	X64
SBB	Reg64，reg64	X64
SBB	rm16,imm8	8086,LOCK
SBB	Rm16，imm8	8086，LOCK
SBB	rm32,imm8	386,LOCK
SBB	Rm32，imm8	386，LOCK
SBB	rm64,imm8	X64,LOCK
SBB	Rm64，imm8	X64，锁定
SBB	reg_al,imm	
SBB	我叫 reg _ al	8086
SBB	reg_ax,sbyteword	8086,ND
SBB	Reg _ ax，字节词	8086，ND
SBB	reg_ax,imm	
SBB	Reg _ ax，imm	8086
SBB	reg_eax,sbytedword	386,ND
SBB	Reg _ eax，bytedword	新界 386 号
SBB	reg_eax,imm	
SBB	注意，注意	386

SBB	reg_rax,sbytedword	X64,ND
SBB	Reg _ rax，字节词	X64，ND
SBB	reg_rax,imm	X64
SBB	Reg _ rax，imm	X64
SBB	rm8,imm	8086,LOCK
SBB	Rm8，imm	8086，LOCK
		8086,LOCK,ND
SBB	rm16,sbyteword	新界洛克 8086
SBB	Rm16，字节词	号

158

SBB	rm16,imm	8086,LOCK
SBB	16 元	8086，LOCK
SBB	rm32,sbytedword	386,LOCK,ND
SBB	Rm32，字节词	新界洛克 386 号
SBB	rm32,imm	386,LOCK
SBB	Rm32，imm	386，LOCK
SBB	rm64,sbytedword	X64,LOCK,ND
SBB	Rm64，字节词	X64，LOCK，ND
SBB	rm64,imm	X64,LOCK
SBB	64 元	X64，锁定
SBB	mem,imm8	8086,LOCK
SBB	Mem，imm8	8086，LOCK
SBB	mem,sbyteword16	8086,LOCK,ND
SBB	Mem，byteword16	新界洛克 8086 号
SBB	mem,imm16	8086,LOCK
SBB	我是 im16	8086，LOCK
SBB	mem,sbytedword32	386,LOCK,ND
SBB	Mem，sbytedword32	新界洛克 386 号
SBB	mem,imm32	386,LOCK
SBB	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
SBB	rm8,imm	8086，LOCK，ND，
SBB	Rm8，imm	NOLONG
SCASB		
SCASB		8086
SCASD		
SCASD		386
SCASQ		X64
SCASQ		X64
SCASW		
SCASW SCASW		8086
SFENCE		X64,AMD
斯芬奇		X64，AMD
SGDT	mem	
SGDT	Mem	286
SHL	rm8,unity	
SHL	8 林吉特，团结	8086
SHL	rm8,reg_cl	
SHL	Rm8，reg _ cl	8086
SHL	rm8,imm8	
SHL	Rm8，imm8	186
SHL	rm16,unity	
SHL	Rm16，团结	8086
SHL	rm16,reg_cl	
SHL	Rm16，reg _ cl	8086
SHL	rm16,imm8	
SHL	Rm16，imm8	186
SHL	rm32,unity	
SHL	Rm32，团结	386
SHL	rm32,reg_cl	386

SHL	Rm32，reg _ cl	
SHL	rm32,imm8	
SHL	Rm32，imm8	386
SHL	rm64,unity	X64
SHL	Rm64，团结	X64
SHL	rm64,reg_cl	X64
SHL	Rm64，reg _ cl	X64
SHL	rm64,imm8	X64
SHL	Rm64，imm8	X64
SHLD	mem,reg16,imm	
SHLD	Mem，reg16，imm	386
SHLD	reg16,reg16,imm	
SHLD	注意，注意，注意，注意	386
SHLD	mem,reg32,imm	
SHLD	第 32 章	386
SHLD	reg32,reg32,imm	
SHLD	Reg32，reg32，imm	386
SHLD	mem,reg64,imm	X64
SHLD	我，reg64，我	X64
SHLD	reg64,reg64,imm	X64
SHLD	Reg64，reg64，imm	X64
SHLD	mem,reg16,reg_cl	
SHLD	Mem，reg16，reg_cl	386
SHLD	reg16,reg16,reg_cl	
SHLD	规则 16，规则 16，规则 cl	386
SHLD	mem,reg32,reg_cl	
SHLD	Mem，reg32，reg_cl	386
SHLD	reg32,reg32,reg_cl	
SHLD	Reg32，reg32，reg_cl	386
SHLD	mem,reg64,reg_cl	X64
SHLD	Mem，reg64，reg_cl	X64
SHLD	reg64,reg64,reg_cl	X64
SHLD	Reg64，reg64，reg_cl	X64
SHR	rm8,unity	
SHR	8 林吉特，团结	8086
SHR	rm8,reg_cl	
SHR	Rm8，reg _ cl	8086
SHR	rm8,imm8	
SHR	Rm8，imm8	186
SHR	rm16,unity	
SHR	Rm16，团结	8086
SHR	rm16,reg_cl	
SHR	Rm16，reg _ cl	8086
SHR	rm16,imm8	
SHR	Rm16，imm8	186
SHR	rm32,unity	
SHR	Rm32，团结	386
SHR	rm32,reg_cl	
SHR	Rm32，reg _ cl	386
SHR	rm32,imm8	386

SHR	Rm32，imm8	
SHR	rm64,unity	X64
SHR	Rm64，团结	X64
SHR	rm64,reg_cl	X64
SHR	Rm64，reg _ cl	X64
SHR	rm64,imm8	X64
SHR	Rm64，imm8	X64
SHRD		
战略人力资源	mem,reg16,imm	
部	Mem，reg16，imm	386

159

SHRD 战略人力资源部	reg16,reg16,imm 注意，注意，注意，注意	386
SHRD 战略人力资源部	mem,reg32,imm 第 32 章	386
SHRD 战略人力资源部	reg32,reg32,imm Reg32，reg32，imm	386
SHRD 战略人力资源部	mem,reg64,imm 我，reg64，我	X64 X64
SHRD 战略人力资源部	reg64,reg64,imm Reg64，reg64，imm	X64 X64
SHRD 战略人力资源部	mem,reg16,reg_cl Mem，reg16，reg_cl	386
SHRD 战略人力资源部	reg16,reg16,reg_cl 规则 16，规则 16，规则 cl	386
SHRD 战略人力资源部	mem,reg32,reg_cl Mem，reg32，reg_cl	386
SHRD 战略人力资源部	reg32,reg32,reg_cl Reg32，reg32，reg_cl	386
SHRD 战略人力资源部	mem,reg64,reg_cl Mem，reg64，reg_cl	X64 X64
SHRD 战略人力资源部	reg64,reg64,reg_cl Reg64，reg64，reg_cl	X64 X64
SIDT	mem	
SIDT	Mem	286
SLDT	mem	
SLDT	Mem	286
SLDT	mem16	
SLDT	Mem16	286
SLDT	reg16	
SLDT	规例 16	286
SLDT	reg32	
SLDT	Reg32	386
SLDT	reg64	X64,ND
SLDT	Reg64	X64，ND
SLDT	reg64	X64
SLDT	Reg64	X64
SKINIT		X64
SKINIT		X64
SMI		386,UNDOC

SMI		386 UNDOC
SMINT		P6,CYRIX,ND
薄荷		P6，CYRIX，ND
SMINTOLD		486,CYRIX,ND,OBSOLETE
斯敏特尔		486，CYRIX，ND，废弃
SMSW	mem	
SMSW	Mem	286
SMSW	mem16	
SMSW	Mem16	286
SMSW	reg16	
SMSW	规例 16	286
SMSW	reg32	
SMSW	Reg32	386
SMSW	reg64	X64
SMSW	Reg64	X64
STC		
STC		8086
STD		
STD 性病		8086
STI		
性传播感染		8086
STOSB		
斯托斯		8086
STOSD		
STOSD		386
STOSQ		X64
STOSQ 斯托克		X64
STOSW		
装载		8086
STR	mem	286,PROT
STR	Mem	286，PROT
STR	mem16	286,PROT
STR	Mem16	286，PROT
STR	reg16	286,PROT
STR	规例 16	286，PROT
STR	reg32	386,PROT
STR	Reg32	386，PROT
STR	reg64	X64
STR	Reg64	X64
SUB	mem,reg8	8086,LOCK
潜艇	Mem reg8	8086，LOCK
SUB	reg8,reg8	
潜艇	Reg8，reg8	8086
SUB	mem,reg16	8086,LOCK
潜艇	Mem，reg16	8086，LOCK
SUB	reg16,reg16	
潜艇	Reg16，reg16	8086
SUB	mem,reg32	386,LOCK
潜艇	Mem reg32	386，LOCK
SUB	reg32,reg32	
潜艇	Reg32，reg32	386

SUB	mem,reg64	X64,LOCK
潜艇	Mem，reg64	X64，锁定
SUB	reg64,reg64	X64
潜艇	Reg64，reg64	X64
SUB	reg8,mem	
潜艇	Reg8，mem	8086
SUB	reg8,reg8	
潜艇	Reg8，reg8	8086
SUB	reg16,mem	
潜艇	Reg16，mem	8086
SUB	reg16,reg16	
潜艇	Reg16，reg16	8086
SUB	reg32,mem	
潜艇	Reg32，mem	386
SUB	reg32,reg32	
潜艇	Reg32，reg32	386
SUB	reg64,mem	X64
潜艇	Reg64，mem	X64

160

SUB	reg64,reg64	X64
潜艇	Reg64，reg64	X64
SUB	rm16,imm8	8086,LOCK
潜艇	Rm16，imm8	8086，LOCK
SUB	rm32,imm8	386,LOCK
潜艇	Rm32，imm8	386，LOCK
SUB	rm64,imm8	X64,LOCK
潜艇	Rm64，imm8	X64，锁定
SUB	reg_al,imm	
潜艇	我叫 reg _ al	8086
SUB	reg_ax,sbyteword	8086,ND
潜艇	Reg _ ax，字节词	8086，ND
SUB	reg_ax,imm	
潜艇	Reg _ ax，imm	8086
SUB	reg_eax,sbytedword	386,ND
潜艇	Reg _ eax，bytedword	新界 386 号
SUB	reg_eax,imm	
潜艇	注意，注意	386
SUB	reg_rax,sbytedword	X64,ND
潜艇	Reg _ rax，字节词	X64，ND
SUB	reg_rax,imm	X64
潜艇	Reg _ rax，imm	X64
SUB	rm8,imm	8086,LOCK
潜艇	Rm8，imm	8086，LOCK
SUB	rm16,sbyteword	8086,LOCK,ND
潜艇	Rm16，字节词	新界洛克 8086 号
SUB	rm16,imm	8086,LOCK
潜艇	16 元	8086，LOCK
SUB	rm32,sbytedword	386,LOCK,ND
潜艇	Rm32，字节词	新界洛克 386 号
SUB	rm32,imm	386,LOCK
潜艇	Rm32，imm	386，LOCK
SUB	rm64,sbytedword	X64,LOCK,ND
潜艇	Rm64，字节词	X64，LOCK，ND
SUB	rm64,imm	X64,LOCK
潜艇	64 元	X64，锁定
SUB	mem,imm8	8086,LOCK
潜艇	Mem，imm8	8086，LOCK
SUB	mem,sbyteword16	8086,LOCK,ND
潜艇	Mem，byteword16	新界洛克 8086 号
SUB	mem,imm16	8086,LOCK
潜艇	我是 im16	8086，LOCK
SUB	mem,sbytedword32	386,LOCK,ND
潜艇	Mem，sbytedword32	新界洛克 386 号
SUB	mem,imm32	386,LOCK
潜艇	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
SUB	rm8,imm	8086，LOCK，ND，
潜艇	Rm8，imm	NOLONG
SVDC	mem80,reg_sreg	486,CYRIX,SMM

SVDC	Mem80，reg _ sreg	486，CYRIX，SMM
SVLDT	mem80	486,CYRIX,SMM,ND
SVLDT	Mem80	486，CYRIX，SMM，ND
SVTS		
SVTS 室上性心	mem80	486,CYRIX,SMM
动过速	Mem80	486，CYRIX，SMM
SWAPGS		X64
交换		X64
SYSCALL		P6,AMD
SYSCALL		P6 AMD
SYSENTER		P6
SYSENTER		P6
SYSEXIT		P6,PRIV
SYSEXIT		P6，PRIV
SYSRET		P6,PRIV,AMD
SYSRET		P6，PRIV，AMD
TEST	mem,reg8	
测试	Mem reg8	8086
TEST	reg8,reg8	
测试	Reg8，reg8	8086
TEST	mem,reg16	
测试	Mem，reg16	8086
TEST	reg16,reg16	
测试	Reg16，reg16	8086
TEST	mem,reg32	
测试	Mem reg32	386
TEST	reg32,reg32	
测试	Reg32，reg32	386
TEST	mem,reg64	X64
测试	Mem，reg64	X64
TEST	reg64,reg64	X64
测试	Reg64，reg64	X64
TEST	reg8,mem	
测试	Reg8，mem	8086
TEST	reg16,mem	
测试	Reg16，mem	8086
TEST	reg32,mem	
测试	Reg32，mem	386
TEST	reg64,mem	X64
测试	Reg64，mem	X64
TEST	reg_al,imm	
测试	我叫 reg _ al	8086
TEST	reg_ax,imm	
测试	Reg _ ax，imm	8086
TEST	reg_eax,imm	
测试	注意，注意	386
TEST	reg_rax,imm	X64
测试	Reg _ rax，imm	X64
TEST	rm8,imm	
测试	Rm8，imm	8086

TEST	rm16,imm	
测试	16 元	8086
TEST	rm32,imm	
测试	Rm32，imm	386
TEST	rm64,imm	X64
测试	64 元	X64
TEST	mem,imm8	
测试	Mem，imm8	8086
TEST	mem,imm16	
测试	我是 im16	8086

161

TEST	mem,imm32	
测试	我 32 岁	386
UD0		186,OBSOLETE
UD0		186，淘汰
UD0	reg16,rm16	
UD0	正版 16，rm16	186
UD0	reg32,rm32	
UD0	Reg32，rm32	186
UD0	reg64,rm64	
UD0	Reg64，rm64	186
UD1	reg,rm16	
UD1	注册，rm16	186
UD1	reg,rm32	
UD1	注册，rm32	186
UD1	reg,rm64	
UD1	注册，rm64	186
UD1		186,ND
UD1		186，ND
UD2B		186,ND
UD2B		186，ND
UD2B	reg,rm16	186,ND
UD2B	注册，rm16	186，ND
UD2B	reg,rm32	186,ND
UD2B	注册，rm32	186，ND
UD2B	reg,rm64	186,ND
UD2B	注册，rm64	186，ND
UD2		
UD2		186
UD2A		186,ND
UD2A		186，ND
UMOV	mem,reg8	386,UNDOC,ND
UMOV	Mem reg8	386 UNDOC ND
UMOV	reg8,reg8	386,UNDOC,ND
UMOV	Reg8，reg8	386 UNDOC ND
UMOV	mem,reg16	386,UNDOC,ND
UMOV	Mem，reg16	386 UNDOC ND
UMOV	reg16,reg16	386,UNDOC,ND
UMOV	Reg16，reg16	386 UNDOC ND
UMOV	mem,reg32	386,UNDOC,ND
UMOV	Mem reg32	386 UNDOC ND
UMOV	reg32,reg32	386,UNDOC,ND
UMOV	Reg32，reg32	386 UNDOC ND
UMOV	reg8,mem	386,UNDOC,ND
UMOV	Reg8，mem	386 UNDOC ND
UMOV	reg8,reg8	386,UNDOC,ND
UMOV	Reg8，reg8	386 UNDOC ND
UMOV	reg16,mem	386,UNDOC,ND
UMOV	Reg16，mem	386 UNDOC ND
UMOV	reg16,reg16	386,UNDOC,ND
UMOV	Reg16，reg16	386 UNDOC ND

UMOV	reg32,mem	386,UNDOC,ND
UMOV	Reg32， mem	386 UNDOC ND
UMOV	reg32,reg32	386,UNDOC,ND
UMOV	Reg32， reg32	386 UNDOC ND
VERR	mem	286,PROT
VERR	Mem	286， PROT
VERR	mem16	286,PROT
VERR	Mem16	286， PROT
VERR	reg16	286,PROT
VERR	规例 16	286， PROT
VERW	mem	286,PROT
威尔士	Mem	286， PROT
VERW	mem16	286,PROT
威尔士	Mem16	286， PROT
VERW	reg16	286,PROT
威尔士	规例 16	286， PROT
FWAIT		
等等		8086
WBINVD		486,PRIV
WBINVD		486 号， PRIV
WRSHR	rm32	P6,CYRIX,SMM
WRSHR	Rm32	P6， CYRIX， SMM
WRMSR		PENT,PRIV
WRMSR		废弃的，私立学校
XADD	mem,reg8	486,LOCK
XADD	Mem reg8	486， 锁定
XADD	reg8,reg8	
XADD	Reg8， reg8	486
XADD	mem,reg16	486,LOCK
XADD	Mem， reg16	486， 锁定
XADD	reg16,reg16	
XADD	Reg16， reg16	486
XADD	mem,reg32	486,LOCK
XADD	Mem reg32	486， 锁定
XADD	reg32,reg32	
XADD	Reg32， reg32	486
XADD	mem,reg64	X64,LOCK
XADD	Mem， reg64	X64， 锁定
XADD	reg64,reg64	X64
XADD	Reg64， reg64	X64
		386,SW,UNDOC,ND
XBTS	reg16,mem	386， SW，
XBTS	Reg16， mem	UNDOC， ND
XBTS	reg16,reg16	386,UNDOC,ND
XBTS	Reg16， reg16	386 UNDOC ND
		386,SD,UNDOC,ND
XBTS	reg32,mem	386， SD，
XBTS	Reg32， mem	UNDOC， ND
XBTS	reg32,reg32	386,UNDOC,ND
XBTS	Reg32， reg32	386 UNDOC ND

XCHG	reg_ax,reg16	
XCHG	Reg_ax， reg16	8086
XCHG	reg_eax,reg32na	
XCHG	Reg_eax， reg32na	386
XCHG	reg_rax,reg64	X64
XCHG	Reg _ rax， reg64	X64
XCHG	reg16,reg_ax	
XCHG	Reg16， reg_ax	8086
XCHG	reg32na,reg_eax	
XCHG	Reg32na， reg_eax	386

162

XCHG	reg64,reg_rax	X64
XCHG	Reg64，reg_rax	X64
XCHG	reg_eax,reg_eax	386,NOLONG
XCHG	Reg _ eax，reg _ eax	386，NOLONG
XCHG	reg8,mem	8086,LOCK
XCHG	Reg8，mem	8086，LOCK
XCHG	reg8,reg8	
XCHG	Reg8，reg8	8086
XCHG	reg16,mem	8086,LOCK
XCHG	Reg16，mem	8086，LOCK
XCHG	reg16,reg16	
XCHG	Reg16，reg16	8086
XCHG	reg32,mem	386,LOCK
XCHG	Reg32，mem	386，LOCK
XCHG	reg32,reg32	
XCHG	Reg32，reg32	386
XCHG	reg64,mem	X64,LOCK
XCHG	Reg64，mem	X64，锁定
XCHG	reg64,reg64	X64
XCHG	Reg64，reg64	X64
XCHG	mem,reg8	8086,LOCK
XCHG	Mem reg8	8086，LOCK
XCHG	reg8,reg8	
XCHG	Reg8，reg8	8086
XCHG	mem,reg16	8086,LOCK
XCHG	Mem，reg16	8086，LOCK
XCHG	reg16,reg16	
XCHG	Reg16，reg16	8086
XCHG	mem,reg32	386,LOCK
XCHG	Mem reg32	386，LOCK
XCHG	reg32,reg32	
XCHG	Reg32，reg32	386
XCHG	mem,reg64	X64,LOCK
XCHG	Mem，reg64	X64，锁定
XCHG	reg64,reg64	X64
XCHG	Reg64，reg64	X64
XLATB		
XLATB		8086
XLAT		
XLAT		8086
XOR	mem,reg8	8086,LOCK
异或	Mem reg8	8086，LOCK
XOR	reg8,reg8	
异或	Reg8，reg8	8086
XOR	mem,reg16	8086,LOCK
异或	Mem，reg16	8086，LOCK
XOR	reg16,reg16	
异或	Reg16，reg16	8086
XOR	mem,reg32	386,LOCK
异或	Mem reg32	386，LOCK

XOR	reg32,reg32	
异或	Reg32，reg32	386
XOR	mem,reg64	X64,LOCK
异或	Mem，reg64	X64，锁定
XOR	reg64,reg64	X64
异或	Reg64，reg64	X64
XOR	reg8,mem	
异或	Reg8，mem	8086
XOR	reg8,reg8	
异或	Reg8，reg8	8086
XOR	reg16,mem	
异或	Reg16，mem	8086
XOR	reg16,reg16	
异或	Reg16，reg16	8086
XOR	reg32,mem	
异或	Reg32，mem	386
XOR	reg32,reg32	
异或	Reg32，reg32	386
XOR	reg64,mem	X64
异或	Reg64，mem	X64
XOR	reg64,reg64	X64
异或	Reg64，reg64	X64
XOR	rm16,imm8	8086,LOCK
异或	Rm16，imm8	8086，LOCK
XOR	rm32,imm8	386,LOCK
异或	Rm32，imm8	386，LOCK
XOR	rm64,imm8	X64,LOCK
异或	Rm64，imm8	X64，锁定
XOR	reg_al,imm	
异或	我叫 reg _ al	8086
XOR	reg_ax,sbyteword	8086,ND
异或	Reg _ ax，字节词	8086，ND
XOR	reg_ax,imm	
异或	Reg _ ax，imm	8086
XOR	reg_eax,sbytedword	386,ND
异或	Reg _ eax，bytedword	新界 386 号
XOR	reg_eax,imm	
异或	注意，注意	386
XOR	reg_rax,sbytedword	X64,ND
异或	Reg _ rax，字节词	X64，ND
XOR	reg_rax,imm	X64
异或	Reg _ rax，imm	X64
XOR	rm8,imm	8086,LOCK
异或	Rm8，imm	8086，LOCK
		8086,LOCK,ND
XOR	rm16,sbyteword	新界洛克 8086
异或	Rm16，字节词	号
XOR	rm16,imm	8086,LOCK
异或	16 元	8086，LOCK
XOR	rm32,sbytedword	386,LOCK,ND

异或	Rm32，字节词	新界洛克 386 号
XOR	rm32,imm	386,LOCK
异或	Rm32，imm	386，LOCK
		X64,LOCK,ND
XOR	rm64,sbytedword	X64，LOCK，
异或	Rm64，字节词	ND
XOR	rm64,imm	X64,LOCK
异或	64 元	X64，锁定
XOR	mem,imm8	8086,LOCK
异或	Mem，imm8	8086，LOCK

163

XOR	mem,sbyteword16	8086,LOCK,ND
异或	Mem，byteword16	新界洛克 8086 号
XOR	mem,imm16	8086,LOCK
异或	我是 im16	8086，LOCK
XOR	mem,sbytedword32	386,LOCK,ND
异或	Mem，sbytedword32	新界洛克 386 号
XOR	mem,imm32	386,LOCK
异或	我 32 岁	386，LOCK
		8086,LOCK,ND,NOLONG
XOR	rm8,imm	8086，LOCK，ND，
异或	Rm8，imm	NOLONG
CMOVcc	reg16,mem	P6
CMOVcc	Reg16，mem	P6
CMOVcc	reg16,reg16	P6
CMOVcc	Reg16，reg16	P6
CMOVcc	reg32,mem	P6
CMOVcc	Reg32，mem	P6
CMOVcc	reg32,reg32	P6
CMOVcc	Reg32，reg32	P6
CMOVcc	reg64,mem	X64
CMOVcc	Reg64，mem	X64
CMOVcc	reg64,reg64	X64
CMOVcc	Reg64，reg64	X64
Jcc	imm|near	386,BND
Jcc	在附近	386，德国联邦情报局
Jcc	imm16|near	386,NOLONG,BND
Jcc	Im16 | 近	386，NOLONG，BND
Jcc	imm32|near	386,NOLONG,BND
Jcc	Im32 | 近	386，NOLONG，BND
Jcc	imm64|near	X64,BND
Jcc	Imm64 | 近	X64，BND
Jcc	imm|short	8086,ND,BND
Jcc	我很短	8086，ND，BND
Jcc	imm	8086,ND,BND
Jcc	是的	8086，ND，BND
Jcc	imm	386,ND,BND
Jcc	是的	386，ND，BND
Jcc	imm	8086,ND,BND
Jcc	是的	8086，ND，BND
Jcc	imm	8086,BND
Jcc	是的	8086，BND
SETcc	mem	
赛特克	Mem	386
SETcc	reg8	
赛特克	Reg8	386

B.1.3 Katmai Streaming SIMD instructions (SSE — a.k.a. KNI, XMM, MMX2)
B. 1.3 Katmai 流 SIMD 指令(SSE-- 又名 KNI，XMM，MMX2)

ADDPS	xmmreg,xmmrm128	KATMAI,SSE
ADDPS	Xmmreg，xmmrm128	KATMAI，SSE

ADDSS	xmmreg,xmmrm32	KATMAI,SSE
地址系统	Xmmrm32	KATMAI，SSE
ANDNPS	xmmreg,xmmrm128	KATMAI,SSE
ANDNPS	Xmmreg，xmmrm128	KATMAI，SSE
ANDPS	xmmreg,xmmrm128	KATMAI,SSE
ANDPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPEQPS	xmmreg,xmmrm128	KATMAI,SSE
CMPEQPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPEQSS	xmmreg,xmmrm32	KATMAI,SSE
CMPEQSS	Xmmrm32	KATMAI，SSE
CMPLEPS	xmmreg,xmmrm128	KATMAI,SSE
CMPLEPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPLESS	xmmreg,xmmrm32	KATMAI,SSE
CMPLESS	Xmmrm32	KATMAI，SSE
CMPLTPS	xmmreg,xmmrm128	KATMAI,SSE
CMPLTPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPLTSS	xmmreg,xmmrm32	KATMAI,SSE
CMPLTSS	Xmmrm32	KATMAI，SSE
CMPNEQPS	xmmreg,xmmrm128	KATMAI,SSE
CMPNEQPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPNEQSS	xmmreg,xmmrm32	KATMAI,SSE
CMPNEQSS	Xmmrm32	KATMAI，SSE
CMPNLEPS	xmmreg,xmmrm128	KATMAI,SSE
CMPNLEPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPNLESS	xmmreg,xmmrm32	KATMAI,SSE
CMPNLESS	Xmmrm32	KATMAI，SSE
CMPNLTPS	xmmreg,xmmrm128	KATMAI,SSE
CMPNLTPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPNLTSS	xmmreg,xmmrm32	KATMAI,SSE
CMPNLTSS	Xmmrm32	KATMAI，SSE
CMPORDPS	xmmreg,xmmrm128	KATMAI,SSE
CMPORDPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPORDSS	xmmreg,xmmrm32	KATMAI,SSE
中央司令部	Xmmrm32	KATMAI，SSE
CMPUNORDPS	xmmreg,xmmrm128	KATMAI,SSE
CMPUNORDPS	Xmmreg，xmmrm128	KATMAI，SSE
CMPUNORDSS	xmmreg,xmmrm32	KATMAI,SSE
CMPUNORDSS	Xmmrm32	KATMAI，SSE
CMPPS	xmmreg,mem,imm	KATMAI,SSE
CMPPS	Xmmreg，mem，imm	KATMAI，SSE
CMPPS	xmmreg,xmmreg,imm	KATMAI,SSE
CMPPS	Xmmreg，xmmreg，imm	KATMAI，SSE
CMPSS	xmmreg,mem,imm	KATMAI,SSE
CMPSS	Xmmreg，mem，imm	KATMAI，SSE
CMPSS	xmmreg,xmmreg,imm	KATMAI,SSE
CMPSS	Xmmreg，xmmreg，imm	KATMAI，SSE
COMISS	xmmreg,xmmrm32	KATMAI,SSE
委员会	Xmmrm32	KATMAI，SSE
CVTPI2PS	xmmreg,mmxrm64	KATMAI,SSE,MMX
CVTPI2PS	Xmmreg，mmxrm64	KATMAI，SSE，MMX

CVTPS2PI	mmxreg,xmmrm64	KATMAI,SSE,MMX
CVTPS2PI	Mmxreg，xmmrm64	KATMAI，SSE，MMX
		KATMAI,SSE,SD,AR1,ND
CVTSI2SS	xmmreg,mem	KATMAI，SSE，SD，
CVTSI2SS	Xmmreg，mem	AR1，ND
CVTSI2SS	xmmreg,rm32	KATMAI,SSE,SD,AR1
CVTSI2SS	Xmmreg，rm32	KATMAI，SSE，SD，AR1

164

CVTSI2SS	xmmreg,rm64	X64,SSE,AR1
CVTSI2SS	Xmmreg，rm64	X64，SSE，AR1
		KATMAI,SSE,SD,AR1
CVTSS2SI	reg32,xmmreg	KATMAI，SSE，SD，
CVTSS2SI	Reg32，xmmreg	AR1
		KATMAI,SSE,SD,AR1
CVTSS2SI	reg32,mem	KATMAI，SSE，SD，
CVTSS2SI	Reg32，mem	AR1
CVTSS2SI	reg64,xmmreg	X64,SSE,SD,AR1
CVTSS2SI	Reg64，xmmreg	X64，SSE，SD，AR1
CVTSS2SI	reg64,mem	X64,SSE,SD,AR1
CVTSS2SI	Reg64，mem	X64，SSE，SD，AR1
CVTTPS2PI	mmxreg,xmmrm	KATMAI,SSE,MMX
CVTTPS2PI	Mmxreg，xmmrm	KATMAI，SSE，MMX
		KATMAI,SSE,SD,AR1
CVTTSS2SI	reg32,xmmrm	KATMAI，SSE，SD，
CVTTSS2SI	Reg32，xmmrm	AR1
CVTTSS2SI	reg64,xmmrm	X64,SSE,SD,AR1
CVTTSS2SI	Reg64，xmmrm	X64，SSE，SD，AR1
DIVPS	xmmreg,xmmrm128	KATMAI,SSE
DIVPS	Xmmreg，xmmrm128	KATMAI，SSE
DIVSS	xmmreg,xmmrm32	KATMAI,SSE
DIVSS	Xmmrm32	KATMAI，SSE
LDMXCSR	mem32	KATMAI,SSE
LDMXCSR	Mem32	KATMAI，SSE
MAXPS	xmmreg,xmmrm128	KATMAI,SSE
MAXPS	Xmmreg，xmmrm128	KATMAI，SSE
MAXSS	xmmreg,xmmrm32	KATMAI,SSE
MAXSS	Xmmrm32	KATMAI，SSE
MINPS	xmmreg,xmmrm128	KATMAI,SSE
MINPS	Xmmreg，xmmrm128	KATMAI，SSE
MINSS	xmmreg,xmmrm32	KATMAI,SSE
MINSS 明斯	Xmmrm32	KATMAI，SSE
MOVAPS	xmmreg,xmmrm128	KATMAI,SSE
MOVAPS	Xmmreg，xmmrm128	KATMAI，SSE
MOVAPS	xmmrm128,xmmreg	KATMAI,SSE
MOVAPS	Xmmrm128，xmreg	KATMAI，SSE
MOVHPS	xmmreg,mem64	KATMAI,SSE
MOVHPS	Xmmreg，mem64	KATMAI，SSE
MOVHPS	mem64,xmmreg	KATMAI,SSE
MOVHPS	Mem64，xmmreg	KATMAI，SSE
MOVLHPS	xmmreg,xmmreg	KATMAI,SSE
移动	Xmmreg，xmmreg	KATMAI，SSE
MOVLPS	xmmreg,mem64	KATMAI,SSE
MOVLPS	Xmmreg，mem64	KATMAI，SSE
MOVLPS	mem64,xmmreg	KATMAI,SSE
MOVLPS	Mem64，xmmreg	KATMAI，SSE
MOVHLPS	xmmreg,xmmreg	KATMAI,SSE
MOVHLPS	Xmmreg，xmmreg	KATMAI，SSE
MOVMSKPS	reg32,xmmreg	KATMAI,SSE

MOVMSKPS	Reg32，xmmreg	KATMAI，SSE
MOVMSKPS	reg64,xmmreg	X64,SSE
MOVMSKPS	Reg64，xmmreg	X64，SSE
MOVNTPS	mem128,xmmreg	KATMAI,SSE
MOVNTPS	Mem128，xmmreg	KATMAI，SSE
MOVSS	xmmreg,xmmrm32	KATMAI,SSE
MOVSS	Xmmrm32	KATMAI，SSE
MOVSS	mem32,xmmreg	KATMAI,SSE
MOVSS	Mem32，xmmreg	KATMAI，SSE
MOVSS	xmmreg,xmmreg	KATMAI,SSE
MOVSS	Xmmreg，xmmreg	KATMAI，SSE
MOVUPS	xmmreg,xmmrm128	KATMAI,SSE
MOVUPS 动作	Xmmreg，xmmrm128	KATMAI，SSE
MOVUPS	xmmrm128,xmmreg	KATMAI,SSE
MOVUPS 动作	Xmmrm128，xmreg	KATMAI，SSE
MULPS	xmmreg,xmmrm128	KATMAI,SSE
MULPS	Xmmreg，xmmrm128	KATMAI，SSE
MULSS	xmmreg,xmmrm32	KATMAI,SSE
马尔斯	Xmmrm32	KATMAI，SSE
ORPS	xmmreg,xmmrm128	KATMAI,SSE
ORPS	Xmmreg，xmmrm128	KATMAI，SSE
RCPPS		
加拿大皇家骑警	xmmreg,xmmrm128	KATMAI,SSE
队	Xmmreg，xmmrm128	KATMAI，SSE
RCPSS	xmmreg,xmmrm32	KATMAI,SSE
RCPSS	Xmmrm32	KATMAI，SSE
RSQRTPS	xmmreg,xmmrm128	KATMAI,SSE
RSQRTPS	Xmmreg，xmmrm128	KATMAI，SSE
RSQRTSS	xmmreg,xmmrm32	KATMAI,SSE
RSQRTSS	Xmmrm32	KATMAI，SSE
	xmmreg,xmmrm128,imm8	
SHUFPS	Xmmreg，xmmmrm128，	KATMAI,SSE
SHUFPS	imm8	KATMAI，SSE
SQRTPS		
长者生活津贴计	xmmreg,xmmrm128	KATMAI,SSE
划	Xmmreg，xmmrm128	KATMAI，SSE
SQRTSS	xmmreg,xmmrm32	KATMAI,SSE
SQRTSS	Xmmrm32	KATMAI，SSE
STMXCSR	mem32	KATMAI,SSE
STMXCSR	Mem32	KATMAI，SSE
SUBPS	xmmreg,xmmrm128	KATMAI,SSE
SUBPS 潜水员	Xmmreg，xmmrm128	KATMAI，SSE
SUBSS	xmmreg,xmmrm32	KATMAI,SSE
潜水艇	Xmmrm32	KATMAI，SSE
UCOMISS	xmmreg,xmmrm32	KATMAI,SSE
UCOMISS	Xmmrm32	KATMAI，SSE
UNPCKHPS	xmmreg,xmmrm128	KATMAI,SSE
UNPCKHPS	Xmmreg，xmmrm128	KATMAI，SSE
UNPCKLPS	xmmreg,xmmrm128	KATMAI,SSE
UNPCKLPS	Xmmreg，xmmrm128	KATMAI，SSE

```
XORPS               xmmreg,xmmrm128               KATMAI,SSE
XORPS               Xmmreg，xmmrm128              KATMAI，SSE
```

B.1.4 Introduced in Deschutes but necessary for SSE support
B. 1.4 引入 Deschutes，但对 SSE 支持是必要的

```
FXRSTOR          mem                      P6,SSE,FPU
FXRSTOR 记忆体 P6，SSE，FPU
FXRSTOR64        mem                      X64,SSE,FPU
FXRSTOR64 mem X64，SSE，FPU
```

```
FXSAVE          mem                     P6,SSE,FPU
FXSAVE mem P6, SSE, FPU
FXSAVE64        mem                     X64,SSE,FPU
FXSAVE64 mem X64, SSE, FPU
```

B.1.5 XSAVE group (AVX and extended state)
B. 1.5 XSAVE 组(AVX 和扩展状态)

XGETBV		NEHALEM
XGETBV		NEHALEM
		NEHALEM,PRIV
XSETBV		NEHALEM，
XSETBV		PRIV
XSAVE	mem	NEHALEM
XSAVE	Mem	NEHALEM
		LONG,NEHALE M
XSAVE64	mem	LONG，
XSAVE64	Mem	NEHALEM
XSAVEC	mem	
XSAVEC	Mem	
XSAVEC64	mem	LONG
XSAVEC64	Mem	龙
XSAVEOPT	mem	
XSAVEOPT	Mem	
XSAVEOPT64	mem	LONG
XSAVEOPT64	Mem	龙
XSAVES	mem	
Xsave	Mem	
XSAVES64	mem	LONG
XSAVES64	Mem	龙
XRSTOR		
XRSTOR x 射线存储器	mem	NEHALEM
	Mem	NEHALEM
		LONG,NEHALE M
XRSTOR64	mem	LONG，
XRSTOR64	Mem	NEHALEM
XRSTORS		
XRSTORS x 射线晶体管	mem	
	Mem	
XRSTORS64	mem	LONG
XRSTORS64	Mem	龙

B.1.6 Generic memory operations
B. 1.6 通用内存操作

PREFETCHNTA	mem8	KATMAI
PREFETCHNTA	Mem8	KATMAI
PREFETCHT0	mem8	KATMAI
PREFETCHT0	Mem8	KATMAI
PREFETCHT1	mem8	KATMAI
PREFETCHT1	Mem8	KATMAI

PREFETCHT2	mem8	KATMAI
PREFETCHT2	Mem8	KATMAI
SFENCE		KATMAI
斯芬奇		KATMAI

B.1.7 New MMX instructions introduced in Katmai
B. 1.7 Katmai 中引入的新 MMX 指令

MASKMOVQ	mmxreg,mmxreg	KATMAI,MMX
MASKMOVQ	Mmxreg，mmxreg	卡特迈，MMX
MOVNTQ	mem,mmxreg	KATMAI,MMX
MOVNTQ	嗯，mmxreg	卡特迈，MMX
PAVGB	mmxreg,mmxrm	KATMAI,MMX
帕夫格勃	Mmxreg mmxrm	卡特迈，MMX
PAVGW	mmxreg,mmxrm	KATMAI,MMX
PAVGW	Mmxreg mmxrm	卡特迈，MMX
PEXTRW	reg32,mmxreg,imm	KATMAI,MMX
PEXTRW	Reg32，mmxreg，imm	卡特迈，MMX
PINSRW	mmxreg,mem,imm	KATMAI,MMX
PINSRW	我，我，我	卡特迈，MMX
PINSRW	mmxreg,rm16,imm	KATMAI,MMX
PINSRW	Mmxreg，rm16，imm	卡特迈，MMX
PINSRW	mmxreg,reg32,imm	KATMAI,MMX
PINSRW	Mmxreg，reg32，imm	卡特迈，MMX
PMAXSW	mmxreg,mmxrm	KATMAI,MMX
PMAXSW	Mmxreg mmxrm	卡特迈，MMX
PMAXUB	mmxreg,mmxrm	KATMAI,MMX
PMAXUB	Mmxreg mmxrm	卡特迈，MMX
PMINSW	mmxreg,mmxrm	KATMAI,MMX
PMINSW	Mmxreg mmxrm	卡特迈，MMX

PMINUB	mmxreg,mmxrm	KATMAI,MMX
PMINUB	Mmxreg mmxrm	卡特迈，MMX
PMOVMSKB	reg32,mmxreg	KATMAI,MMX
PMOVMSKB	Reg32，mmxreg	卡特迈，MMX
PMULHUW	mmxreg,mmxrm	KATMAI,MMX
PMULHUW	Mmxreg mmxrm	卡特迈，MMX
PSADBW	mmxreg,mmxrm	KATMAI,MMX
PSADBW	Mmxreg mmxrm	卡特迈，MMX
PSHUFW	mmxreg,mmxrm,imm	KATMAI,MMX
PSHUFW	Mmxreg，mmxrm，imm	卡特迈，MMX

B.1.8 AMD Enhanced 3DNow! (Athlon) instructions

B. 1.8 AMD 增强型 3dnow (Athlon)指令

PF2IW	mmxreg,mmxrm	PENT,3DNOW
PF2IW	Mmxreg mmxrm	废弃，3DNOW
PFNACC	mmxreg,mmxrm	PENT,3DNOW
Pnacc	Mmxreg mmxrm	废弃，3DNOW
PFPNACC	mmxreg,mmxrm	PENT,3DNOW
PFPNACC	Mmxreg mmxrm	废弃，3DNOW
PI2FW	mmxreg,mmxrm	PENT,3DNOW
PI2FW	Mmxreg mmxrm	废弃，3DNOW
PSWAPD	mmxreg,mmxrm	PENT,3DNOW
PSWAPD	Mmxreg mmxrm	废弃，3DNOW

B.1.9 Willamette SSE2 Cacheability Instructions
B. 1.9 Willamette sse2 可缓存性说明

MASKMOVDQU	xmmreg,xmmreg	WILLAMETTE,SSE2
MASKMOVDQU	Xmmreg，xmmreg	WILLAMETTE，SSE2
CLFLUSH	mem	WILLAMETTE,SSE2
CLFLUSH	Mem	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MOVNTDQ	mem,xmmreg	WILLAMETTE，SSE2，
MOVNTDQ	我的妈妈，妈妈	SO
MOVNTI	mem,reg32	WILLAMETTE,SD
MOVNTI	Mem reg32	WILLAMETTE，SD
MOVNTI	mem,reg64	X64
MOVNTI	Mem，reg64	X64
		WILLAMETTE,SSE2,SO
MOVNTPD	mem,xmmreg	WILLAMETTE，SSE2，
MOVNTPD	我的妈妈，妈妈	SO
LFENCE		WILLAMETTE,SSE2
LFENCE		WILLAMETTE，SSE2
MFENCE		WILLAMETTE,SSE2
MFENCE		WILLAMETTE，SSE2

B.1.10 Willamette MMX instructions (SSE2 SIMD Integer Instructions)
B. 1.10 Willamette MMX 指令(SSE2 SIMD Integer 指令)

		WILLAMETTE,SSE2,SD
MOVD	mem,xmmreg	WILLAMETTE，SSE2，
MOVD	我的妈妈，妈妈	SD
		WILLAMETTE,SSE2,SD
MOVD	xmmreg,mem	WILLAMETTE，SSE2，
MOVD	Xmmreg，mem	SD
MOVD	xmmreg,rm32	WILLAMETTE,SSE2
MOVD	Xmmreg，rm32	WILLAMETTE，SSE2
MOVD	rm32,xmmreg	WILLAMETTE,SSE2
MOVD	Rm32，xmmreg	WILLAMETTE，SSE2
MOVDQA	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVDQA	Xmmreg，xmmreg	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MOVDQA	mem,xmmreg	WILLAMETTE，SSE2，
MOVDQA	我的妈妈，妈妈	SO
		WILLAMETTE,SSE2,SO
MOVDQA	xmmreg,mem	WILLAMETTE，SSE2，
MOVDQA	Xmmreg，mem	SO
MOVDQA	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVDQA	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVDQU	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVDQU	Xmmreg，xmmreg	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MOVDQU	mem,xmmreg	WILLAMETTE，SSE2，
MOVDQU	我的妈妈，妈妈	SO
MOVDQU	xmmreg,mem	WILLAMETTE,SSE2,SO
MOVDQU	Xmmreg，mem	WILLAMETTE，SSE2，

		SO
MOVDQU	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVDQU	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVDQ2Q	mmxreg,xmmreg	WILLAMETTE,SSE2
MOVDQ2Q	Mmxreg，xmmreg	WILLAMETTE，SSE2
MOVQ	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVQ	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVQ	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVQ	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVQ	mem,xmmreg	WILLAMETTE,SSE2
MOVQ	我的妈妈，妈妈	WILLAMETTE，SSE2
MOVQ	xmmreg,mem	WILLAMETTE,SSE2
MOVQ	Xmmreg，mem	WILLAMETTE，SSE2
MOVQ	xmmreg,rm64	X64,SSE2
MOVQ	Xmmreg，rm64	X64，SSE2
MOVQ	rm64,xmmreg	X64,SSE2
MOVQ	Rm64，xmmreg	X64，SSE2
MOVQ2DQ	xmmreg,mmxreg	WILLAMETTE,SSE2
MOVQ2DQ	Xmmreg，mmxreg	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
PACKSSWB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PACKSSWB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PACKSSDW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PACKSSDW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PACKUSWB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PACKUSWB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDW	xmmreg,xmmrm	WILLAMETTE，SSE2，
帕德沃	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDD	Xmmreg，xmmrm	SO
PADDQ	mmxreg,mmxrm	WILLAMETTE,MMX
PADDQ	Mmxreg mmxrm	WILLAMETTE，MMX
		WILLAMETTE,SSE2,SO
PADDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDSB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDSB 帕德斯	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PADDUSB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDUSB	Xmmreg，xmmrm	SO

		WILLAMETTE,SSE2,SO
PADDUSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PADDUSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PAND	xmmreg,xmmrm	WILLAMETTE，SSE2，
潘德	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PANDN	xmmreg,xmmrm	WILLAMETTE，SSE2，
潘登	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PAVGB	xmmreg,xmmrm	WILLAMETTE，SSE2，
帕夫格勃	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PAVGW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PAVGW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPEQB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPEQB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPEQW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPEQW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPEQD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPEQD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPGTB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPGTB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPGTW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPGTW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PCMPGTD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PCMPGTD	Xmmreg，xmmrm	SO

PEXTRW	reg32,xmmreg,imm	WILLAMETTE,SSE2
PEXTRW	Reg32，xmmreg，imm	WILLAMETTE，SSE2
PEXTRW	reg64,xmmreg,imm	X64,SSE2,ND
PEXTRW	Reg64，xmmreg，imm	X64，SSE2，ND
PINSRW	xmmreg,reg16,imm	WILLAMETTE,SSE2
PINSRW	Xmmreg，reg16，imm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,ND
PINSRW	xmmreg,reg32,imm	WILLAMETTE，SSE2，
PINSRW	Xmmreg，reg32，imm	ND
PINSRW	xmmreg,reg64,imm	X64,SSE2,ND
PINSRW	Xmmreg，reg64，imm	X64，SSE2，ND
PINSRW	xmmreg,mem,imm	WILLAMETTE,SSE2
PINSRW	Xmmreg，mem，imm	WILLAMETTE，SSE2
PINSRW	xmmreg,mem16,imm	WILLAMETTE,SSE2
PINSRW	Xmmreg，mem16，imm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
PMADDWD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMADDWD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMAXSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMAXSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMAXUB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMAXUB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMINSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMINSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMINUB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMINUB	Xmmreg，xmmrm	SO
PMOVMSKB	reg32,xmmreg	WILLAMETTE,SSE2
PMOVMSKB	Reg32，xmmreg	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
PMULHUW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMULHUW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMULHW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMULHW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMULLW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMULLW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PMULUDQ	mmxreg,mmxrm	WILLAMETTE，SSE2，
PMULUDQ	Mmxreg mmxrm	SO
		WILLAMETTE,SSE2,SO
PMULUDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PMULUDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
POR	xmmreg,xmmrm	WILLAMETTE，SSE2，
POR	Xmmreg，xmmrm	SO

		WILLAMETTE,SSE2,SO
PSADBW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSADBW	Xmmreg，xmmrm	SO
PSHUFD	xmmreg,xmmreg,imm	WILLAMETTE,SSE2
PSHUFD	Xmmreg，xmmreg，imm	WILLAMETTE，SSE2
PSHUFD	xmmreg,mem,imm	WILLAMETTE,SSE2
PSHUFD	Xmmreg，mem，imm	WILLAMETTE，SSE2
PSHUFHW	xmmreg,xmmreg,imm	WILLAMETTE,SSE2
PSHUFHW	Xmmreg，xmmreg，imm	WILLAMETTE，SSE2
PSHUFHW	xmmreg,mem,imm	WILLAMETTE,SSE2
PSHUFHW	Xmmreg，mem，imm	WILLAMETTE，SSE2
PSHUFLW	xmmreg,xmmreg,imm	WILLAMETTE,SSE2
PSHUFLW	Xmmreg，xmmreg，imm	WILLAMETTE，SSE2
PSHUFLW	xmmreg,mem,imm	WILLAMETTE,SSE2
PSHUFLW	Xmmreg，mem，imm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,AR1
PSLLDQ	xmmreg,imm	WILLAMETTE，SSE2，
PSLLDQ	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSLLW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSLLW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSLLW	xmmreg,imm	WILLAMETTE，SSE2，
PSLLW	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSLLD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSLLD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSLLD	xmmreg,imm	WILLAMETTE，SSE2，
PSLLD	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSLLQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSLLQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSLLQ	xmmreg,imm	WILLAMETTE，SSE2，
PSLLQ	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSRAW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSRAW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSRAW	xmmreg,imm	WILLAMETTE，SSE2，
PSRAW	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSRAD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSRAD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSRAD	xmmreg,imm	WILLAMETTE，SSE2，
PSRAD	快点，快点	AR1
		WILLAMETTE,SSE2,AR1
PSRLDQ	xmmreg,imm	WILLAMETTE，SSE2，
PSRLDQ	快点，快点	AR1

		WILLAMETTE,SSE2,SO
PSRLW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSRLW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSRLW	xmmreg,imm	WILLAMETTE，SSE2，
PSRLW	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSRLD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSRLD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSRLD	xmmreg,imm	WILLAMETTE，SSE2，
PSRLD	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSRLQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSRLQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,AR1
PSRLQ	xmmreg,imm	WILLAMETTE，SSE2，
PSRLQ	快点，快点	AR1
		WILLAMETTE,SSE2,SO
PSUBB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBQ	mmxreg,mmxrm	WILLAMETTE，SSE2，
PSUBQ	Mmxreg mmxrm	SO
		WILLAMETTE,SSE2,SO
PSUBQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBSB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBSB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBUSB	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBUSB	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PSUBUSW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PSUBUSW	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKHBW	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKHBW	Xmmreg，xmmrm	SO

168

		WILLAMETTE,SSE2,SO
PUNPCKHWD	xmmreg,xmmrm	WILLAMETTE，SSE2，
庞普克人	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKHDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKHDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKHQDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKHQDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKLBW	xmmreg,xmmrm	WILLAMETTE，SSE2，
彭彭	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKLWD	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKLWD	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKLDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKLDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PUNPCKLQDQ	xmmreg,xmmrm	WILLAMETTE，SSE2，
PUNPCKLQDQ	Xmmreg，xmmrm	SO
		WILLAMETTE,SSE2,SO
PXOR	xmmreg,xmmrm	WILLAMETTE，SSE2，
PXOR	Xmmreg，xmmrm	SO

B.1.11 Willamette Streaming SIMD instructions (SSE2)
B. 1.11 Willamette 串流 SIMD 指令(SSE2)

ADDPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
多动症警察	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
ADDSD		
ADDSD 注意力缺	xmmreg,xmmrm	WILLAMETTE,SSE2
陷	Xmmreg，xmmrm	WILLAMETTE，SSE2
ANDNPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
ANDNPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
ANDPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
ANDPD 安德森	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPEQPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPEQPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPEQSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPEQSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPLEPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPLEPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPLESD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPLESD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPLTPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPLTPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPLTSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPLTSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPNEQPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPNEQPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO

CMPNEQSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPNEQSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPNLEPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPNLEPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPNLESD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPNLESD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPNLTPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPNLTPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPNLTSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPNLTSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPORDPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPORDPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPORDSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPORDSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CMPUNORDPD	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CMPUNORDPD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CMPUNORDSD	xmmreg,xmmrm	WILLAMETTE,SSE2
CMPUNORDSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
	xmmreg,xmmrm128,imm8	
CMPPD	Xmmreg，xmmmrm128，	WILLAMETTE,SSE2
CMPPD	imm8	WILLAMETTE，SSE2
	xmmreg,xmmrm128,imm8	
CMPSD	Xmmreg，xmmmrm128，	WILLAMETTE,SSE2
CMPSD	imm8	WILLAMETTE，SSE2
COMISD	xmmreg,xmmrm	WILLAMETTE,SSE2
委员会	Xmmreg，xmmrm	WILLAMETTE，SSE2
CVTDQ2PD	xmmreg,xmmrm	WILLAMETTE,SSE2
CVTDQ2PD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CVTDQ2PS	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTDQ2PS	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CVTPD2DQ	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTPD2DQ	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CVTPD2PI	mmxreg,xmmrm	WILLAMETTE,SSE2,SO
CVTPD2PI	Mmxreg，xmmrm	WILLAMETTE，SSE2，SO
CVTPD2PS	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTPD2PS	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CVTPI2PD	xmmreg,mmxrm	WILLAMETTE,SSE2
CVTPI2PD	Xmmreg，mmxrm	WILLAMETTE，SSE2
CVTPS2DQ	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTPS2DQ	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CVTPS2PD	xmmreg,xmmrm	WILLAMETTE,SSE2
CVTPS2PD	Xmmreg，xmmrm	WILLAMETTE，SSE2
CVTSD2SI	reg32,xmmreg	WILLAMETTE,SSE2,AR1
CVTSD2SI	Reg32，xmmreg	WILLAMETTE，SSE2，AR1
CVTSD2SI	reg32,mem	WILLAMETTE,SSE2,AR1
CVTSD2SI	Reg32，mem	WILLAMETTE，SSE2，AR1
CVTSD2SI	reg64,xmmreg	X64,SSE2,AR1
CVTSD2SI	Reg64，xmmreg	X64，SSE2，AR1
CVTSD2SI	reg64,mem	X64,SSE2,AR1
CVTSD2SI	Reg64，mem	X64，SSE2，AR1

CVTSD2SS	xmmreg,xmmrm	WILLAMETTE,SSE2
CVTSD2SS	Xmmreg，xmmrm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SD,AR1,ND
CVTSI2SD	xmmreg,mem	WILLAMETTE，SSE2，SD，
CVTSI2SD	Xmmreg，mem	AR1，ND
CVTSI2SD	xmmreg,rm32	WILLAMETTE,SSE2,SD,AR1
CVTSI2SD	Xmmreg，rm32	WILLAMETTE，SSE2，SD，AR1
CVTSI2SD	xmmreg,rm64	X64,SSE2,AR1
CVTSI2SD	Xmmreg，rm64	X64，SSE2，AR1
CVTSS2SD	xmmreg,xmmrm	WILLAMETTE,SSE2,SD
CVTSS2SD	Xmmreg，xmmrm	WILLAMETTE，SSE2，SD
CVTTPD2PI	mmxreg,xmmrm	WILLAMETTE,SSE2,SO
CVTTPD2PI	Mmxreg，xmmrm	WILLAMETTE，SSE2，SO
CVTTPD2DQ	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTTPD2DQ	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO
CVTTPS2DQ	xmmreg,xmmrm	WILLAMETTE,SSE2,SO
CVTTPS2DQ	Xmmreg，xmmrm	WILLAMETTE，SSE2，SO

169

		WILLAMETTE,SSE2,AR1
CVTTSD2SI	reg32,xmmreg	WILLAMETTE，SSE2，
CVTTSD2SI	Reg32，xmmreg	AR1
		WILLAMETTE,SSE2,AR1
CVTTSD2SI	reg32,mem	WILLAMETTE，SSE2，
CVTTSD2SI	Reg32，mem	AR1
CVTTSD2SI	reg64,xmmreg	X64,SSE2,AR1
CVTTSD2SI	Reg64，xmmreg	X64，SSE2，AR1
CVTTSD2SI	reg64,mem	X64,SSE2,AR1
CVTTSD2SI	Reg64，mem	X64，SSE2，AR1
		WILLAMETTE,SSE2,SO
DIVPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
DIVPD	Xmmreg，xmmrm	SO
DIVSD	xmmreg,xmmrm	WILLAMETTE,SSE2
DIVSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MAXPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
MAXPD	Xmmreg，xmmrm	SO
MAXSD	xmmreg,xmmrm	WILLAMETTE,SSE2
MAXSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MINPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
MINPD	Xmmreg，xmmrm	SO
MINSD	xmmreg,xmmrm	WILLAMETTE,SSE2
MINSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
MOVAPD		
MOVAPD 莫瓦普	xmmreg,xmmreg	WILLAMETTE,SSE2
德	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVAPD		
MOVAPD 莫瓦普	xmmreg,xmmreg	WILLAMETTE,SSE2
德	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVAPD		WILLAMETTE,SSE2,SO
MOVAPD 莫瓦普	mem,xmmreg	WILLAMETTE，SSE2，
德	我的妈妈，妈妈	SO
MOVAPD		WILLAMETTE,SSE2,SO
MOVAPD 莫瓦普	xmmreg,mem	WILLAMETTE，SSE2，
德	Xmmreg，mem	SO
MOVHPD	mem,xmmreg	WILLAMETTE,SSE2
MOVHPD 警察局	我的妈妈，妈妈	WILLAMETTE，SSE2
MOVHPD	xmmreg,mem	WILLAMETTE,SSE2
MOVHPD 警察局	Xmmreg，mem	WILLAMETTE，SSE2
MOVLPD	mem64,xmmreg	WILLAMETTE,SSE2
MOVLPD	Mem64，xmmreg	WILLAMETTE，SSE2
MOVLPD	xmmreg,mem64	WILLAMETTE,SSE2
MOVLPD	Xmmreg，mem64	WILLAMETTE，SSE2
MOVMSKPD	reg32,xmmreg	WILLAMETTE,SSE2
MOVMSKPD	Reg32，xmmreg	WILLAMETTE，SSE2
MOVMSKPD	reg64,xmmreg	X64,SSE2
MOVMSKPD	Reg64，xmmreg	X64，SSE2
MOVSD	xmmreg,xmmreg	WILLAMETTE,SSE2

MOVSD	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVSD	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVSD	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVSD	mem64,xmmreg	WILLAMETTE,SSE2
MOVSD	Mem64，xmmreg	WILLAMETTE，SSE2
MOVSD	xmmreg,mem64	WILLAMETTE,SSE2
MOVSD	Xmmreg，mem64	WILLAMETTE，SSE2
MOVUPD	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVUPD	Xmmreg，xmmreg	WILLAMETTE，SSE2
MOVUPD	xmmreg,xmmreg	WILLAMETTE,SSE2
MOVUPD	Xmmreg，xmmreg	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
MOVUPD	mem,xmmreg	WILLAMETTE，SSE2，
MOVUPD	我的妈妈，妈妈	SO
		WILLAMETTE,SSE2,SO
MOVUPD	xmmreg,mem	WILLAMETTE，SSE2，
MOVUPD	Xmmreg，mem	SO
		WILLAMETTE,SSE2,SO
MULPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
警局	Xmmreg，xmmrm	SO
MULSD	xmmreg,xmmrm	WILLAMETTE,SSE2
MULSD 穆斯德	Xmmreg，xmmrm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
ORPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
ORPD	Xmmreg，xmmrm	SO
SHUFPD	xmmreg,xmmreg,imm	WILLAMETTE,SSE2
SHUFPD	Xmmreg，xmmreg，imm	WILLAMETTE，SSE2
SHUFPD	xmmreg,mem,imm	WILLAMETTE,SSE2
SHUFPD	Xmmreg，mem，imm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
SQRTPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
SQRTPD	Xmmreg，xmmrm	SO
SQRTSD	xmmreg,xmmrm	WILLAMETTE,SSE2
SQRTSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
		WILLAMETTE,SSE2,SO
SUBPD	xmmreg,xmmrm	WILLAMETTE，SSE2，
亚警察局	Xmmreg，xmmrm	SO
SUBSD	xmmreg,xmmrm	WILLAMETTE,SSE2
SUBSD	Xmmreg，xmmrm	WILLAMETTE，SSE2
UCOMISD	xmmreg,xmmrm	WILLAMETTE,SSE2
UCOMISD	Xmmreg，xmmrm	WILLAMETTE，SSE2
UNPCKHPD	xmmreg,xmmrm128	WILLAMETTE,SSE2
UNPCKHPD	Xmmreg，xmmrm128	WILLAMETTE，SSE2
UNPCKLPD	xmmreg,xmmrm128	WILLAMETTE,SSE2
UNPCKLPD	Xmmreg，xmmrm128	WILLAMETTE，SSE2
XORPD	xmmreg,xmmrm128	WILLAMETTE,SSE2
XORPD	Xmmreg，xmmrm128	WILLAMETTE，SSE2

B.1.12 Prescott New Instructions (SSE3)

B. 1.12 Prescott New Instructions (SSE3)

ADDSUBPD	xmmreg,xmmrm	PRESCOTT,SSE3,SO
ADDSUBPD	Xmmreg，xmmrm	PRESCOTT SSE3
ADDSUBPS	xmmreg,xmmrm	PRESCOTT,SSE3,SO
ADDSUBPS 附录	Xmmreg，xmmrm	PRESCOTT SSE3
HADDPD	xmmreg,xmmrm	PRESCOTT,SSE3,SO
HADDPD	Xmmreg，xmmrm	PRESCOTT SSE3
HADDPS	xmmreg,xmmrm	PRESCOTT,SSE3,SO
HADDPS	Xmmreg，xmmrm	PRESCOTT SSE3
HSUBPD	xmmreg,xmmrm	PRESCOTT,SSE3,SO
哈桑警局	Xmmreg，xmmrm	PRESCOTT SSE3
HSUBPS	xmmreg,xmmrm	PRESCOTT,SSE3,SO
HSUBPS	Xmmreg，xmmrm	PRESCOTT SSE3
LDDQU	xmmreg,mem	PRESCOTT,SSE3,SO
LDDQU	Xmmreg，mem	PRESCOTT SSE3
MOVDDUP		
MOVDDUP 动起	xmmreg,xmmrm	PRESCOTT,SSE3
来	Xmmreg，xmmrm	PRESCOTT，SSE3
MOVSHDUP	xmmreg,xmmrm	PRESCOTT,SSE3
动起来	Xmmreg，xmmrm	PRESCOTT，SSE3
MOVSLDUP	xmmreg,xmmrm	PRESCOTT,SSE3
MOVSLDUP	Xmmreg，xmmrm	PRESCOTT，SSE3

170

B.1.13 VMX/SVM Instructions
B. 1.13 VMX/SVM 指令

CLGI		VMX,AMD
CLGI		VMX，AMD
STGI		VMX,AMD
STGI		VMX，AMD
VMCALL		VMX
VMCALL		VMX
VMCLEAR	mem	VMX
VMCLEAR	Mem	VMX
VMFUNC		VMX
VMFUNC		VMX
VMLAUNCH		VMX
VMLAUNCH		VMX
VMLOAD		VMX,AMD
VMLOAD		VMX，AMD
VMMCALL		VMX,AMD
VMMCALL		VMX，AMD
VMPTRLD	mem	VMX
VMPTRLD	Mem	VMX
VMPTRST	mem	VMX
VMPTRST	Mem	VMX
		VMX,NOLONG,SD
VMREAD	rm32,reg32	VMX，
VMREAD	Rm32，reg32	NOLONG，SD
VMREAD	rm64,reg64	X64,VMX
VMREAD	Rm64，reg64	X64 VMX
VMRESUME		
VMRESUME		VMX
VMRESUME		VMX
VMRUN		VMX,AMD
VMRUN		VMX，AMD
VMSAVE		VMX,AMD
VMSAVE		VMX，AMD
		VMX,NOLONG,SD
VMWRITE	reg32,rm32	VMX，
VMWRITE	Reg32，rm32	NOLONG，SD
VMWRITE	reg64,rm64	X64,VMX
VMWRITE	Reg64，rm64	X64 VMX
VMXOFF		
VMXOFF		VMX
VMXOFF		VMX
VMXON	mem	VMX
VMXON	Mem	VMX

B.1.14 Extended Page Tables VMX instructions
B. 1.14 扩展页表 VMX 指令

INVEPT	reg32,mem	VMX,SO,NOLON

投资促进局	Reg32，mem	G
		VMX，SO，
		NOLONG VMX，
		SO，NOLONG
		VMX,SO,LONG
INVEPT	reg64,mem	VMX，SO，
投资促进局	Reg64，mem	LONG
		VMX,SO,NOLON
		G
		VMX，SO，
INVVPID	reg32,mem	NOLONG VMX，
INVVPID	Reg32，mem	SO，NOLONG
		VMX,SO,LONG
INVVPID	reg64,mem	VMX，SO，
INVVPID	Reg64，mem	LONG

B.1.15 Tejas New Instructions (SSSE3)
1.15 Tejas 新指令(SSSE3)

		SSSE3,MMX
PABSB	mmxreg,mmxrm	SSSE3，
PABSB	Mmxreg mmxrm	MMX
PABSB	xmmreg,xmmrm	SSSE3
PABSB	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PABSW	mmxreg,mmxrm	SSSE3，
PABSW	Mmxreg mmxrm	MMX
PABSW	xmmreg,xmmrm	SSSE3
PABSW	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PABSD	mmxreg,mmxrm	SSSE3，
PABSD	Mmxreg mmxrm	MMX
PABSD	xmmreg,xmmrm	SSSE3
PABSD	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PALIGNR	mmxreg,mmxrm,imm	SSSE3，
帕林格尔	Mmxreg，mmxrm，imm	MMX
PALIGNR	xmmreg,xmmrm,imm	SSSE3
帕林格尔	Xmmrg，xmmrm，imm	SSSE3
		SSSE3,MMX
PHADDW	mmxreg,mmxrm	SSSE3，
PHADDW	Mmxreg mmxrm	MMX
PHADDW	xmmreg,xmmrm	SSSE3
PHADDW	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PHADDD	mmxreg,mmxrm	SSSE3，
PHADDD	Mmxreg mmxrm	MMX
PHADDD	xmmreg,xmmrm	SSSE3
PHADDD	Xmmreg，xmmrm	SSSE3
PHADDSW	mmxreg,mmxrm	SSSE3,MMX
PHADDSW	Mmxreg mmxrm	SSSE3，

PHADDSW		MMX
PHADDSW		
PHADDSW	xmmreg,xmmrm	SSSE3
PHADDSW	Xmmreg，xmmrm	SSSE3
PHSUBW		SSSE3,MMX
PHSUBW	mmxreg,mmxrm	SSSE3，
PHSUBW	Mmxreg mmxrm	MMX
PHSUBW		
PHSUBW	xmmreg,xmmrm	SSSE3
PHSUBW	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PHSUBD	mmxreg,mmxrm	SSSE3，
PHSUBD	Mmxreg mmxrm	MMX
PHSUBD	xmmreg,xmmrm	SSSE3
PHSUBD	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PHSUBSW	mmxreg,mmxrm	SSSE3，
PHSUBSW	Mmxreg mmxrm	MMX
PHSUBSW	xmmreg,xmmrm	SSSE3
PHSUBSW	Xmmreg，xmmrm	SSSE3
PMADDUBSW		SSSE3,MMX
PMADDUBSW	mmxreg,mmxrm	SSSE3，
PMADDUBSW	Mmxreg mmxrm	MMX
PMADDUBSW		
PMADDUBSW	xmmreg,xmmrm	SSSE3
PMADDUBSW	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PMULHRSW	mmxreg,mmxrm	SSSE3，
PMULHRSW	Mmxreg mmxrm	MMX
PMULHRSW	xmmreg,xmmrm	SSSE3
PMULHRSW	Xmmreg，xmmrm	SSSE3

171

		SSSE3,MMX
PSHUFB	mmxreg,mmxrm	SSSE3，
PSHUFB	Mmxreg mmxrm	MMX
PSHUFB	xmmreg,xmmrm	SSSE3
PSHUFB	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PSIGNB	mmxreg,mmxrm	SSSE3，
PSIGNB	Mmxreg mmxrm	MMX
PSIGNB	xmmreg,xmmrm	SSSE3
PSIGNB	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PSIGNW	mmxreg,mmxrm	SSSE3，
PSIGNW	Mmxreg mmxrm	MMX
PSIGNW	xmmreg,xmmrm	SSSE3
PSIGNW	Xmmreg，xmmrm	SSSE3
		SSSE3,MMX
PSIGND	mmxreg,mmxrm	SSSE3，
PSIGND	Mmxreg mmxrm	MMX
PSIGND	xmmreg,xmmrm	SSSE3
PSIGND	Xmmreg，xmmrm	SSSE3

B.1.16 AMD SSE4A
乙. 1.16 AMD SSE4A

EXTRQ	xmmreg,imm,imm	SSE4A,AMD
EXTRQ 外援	Xmmreg，im，imm	SSE4A，AMD
EXTRQ	xmmreg,xmmreg	SSE4A,AMD
EXTRQ 外援	Xmmreg，xmmreg	SSE4A，AMD
	xmmreg,xmmreg,imm,imm	
INSERTQ	Xmmreg，xmmreg，imm，	SSE4A,AMD
插入	imm	SSE4A，AMD
INSERTQ	xmmreg,xmmreg	SSE4A,AMD
插入	Xmmreg，xmmreg	SSE4A，AMD
MOVNTSD	mem,xmmreg	SSE4A,AMD
MOVNTSD	我的妈妈，妈妈	SSE4A，AMD
		SSE4A,AMD,SD
MOVNTSS	mem,xmmreg	SSE4A，AMD，
MOVNTSS	我的妈妈，妈妈	SD

B.1.17 New instructions in Barcelona
B. 1.17 巴塞罗那的新指示

LZCNT	reg16,rm16	P6,AMD
LZCNT	正版 16，rm16	P6 AMD
LZCNT	reg32,rm32	P6,AMD
LZCNT	Reg32，rm32	P6 AMD
		X64,AM
		D
LZCNT	reg64,rm64	X64，
LZCNT	Reg64，rm64	AMD

B.1.18 Penryn New Instructions (SSE4.1)

B. 1.18 Penryn 新指令(SSE4.1)

BLENDPD	xmmreg,xmmrm,imm	SSE41
BLENDPD	Xmmrg，xmmrm，imm	SSE41
BLENDPS	xmmreg,xmmrm,imm	SSE41
混合饮料	Xmmrg，xmmrm，imm	SSE41
	xmmreg,xmmrm,xmm0	
BLENDVPD	Xmmreg，xmmmrm，	SSE41
BLENDVPD	xmm0	SSE41
BLENDVPD	xmmreg,xmmrm	SSE41
BLENDVPD	Xmmreg，xmmrm	SSE41
	xmmreg,xmmrm,xmm0	
BLENDVPS	Xmmreg，xmmmrm，	SSE41
BLENDVPS	xmm0	SSE41
BLENDVPS	xmmreg,xmmrm	SSE41
BLENDVPS	Xmmreg，xmmrm	SSE41
DPPD	xmmreg,xmmrm,imm	SSE41
私隐专员公署	Xmmrg，xmmrm，imm	SSE41
DPPS	xmmreg,xmmrm,imm	SSE41
DPPS	Xmmrg，xmmrm，imm	SSE41
EXTRACTPS	rm32,xmmreg,imm	SSE41
提取物	Rm32，xmmreg，imm	SSE41
		SSE41,X64
EXTRACTPS	reg64,xmmreg,imm	SSE41，
提取物	Reg64，xmmreg，imm	X64
INSERTPS	xmmreg,xmmrm,imm	SSE41,SD
插入	Xmmrg，xmmrm，imm	SSE41，SD
MOVNTDQA	xmmreg,mem128	SSE41
MOVNTDQA	Xmmreg，mem128	SSE41
MPSADBW	xmmreg,xmmrm,imm	SSE41
MPSADBW	Xmmrg，xmmrm，imm	SSE41
PACKUSDW	xmmreg,xmmrm	SSE41
PACKUSDW	Xmmreg，xmmrm	SSE41
	xmmreg,xmmrm,xmm0	
PBLENDVB	Xmmreg，xmmmrm，	SSE41
PBLENDVB	xmm0	SSE41
PBLENDVB	xmmreg,xmmrm	SSE41
PBLENDVB	Xmmreg，xmmrm	SSE41
PBLENDW	xmmreg,xmmrm,imm	SSE41
PBLENDW	Xmmrg，xmmrm，imm	SSE41
PCMPEQQ	xmmreg,xmmrm	SSE41
PCMPEQQ	Xmmreg，xmmrm	SSE41
PEXTRB	reg32,xmmreg,imm	SSE41
PEXTRB	Reg32，xmmreg，imm	SSE41
PEXTRB	mem8,xmmreg,imm	SSE41
PEXTRB	Mem8，xmmreg，imm	SSE41
		SSE41,X64
PEXTRB	reg64,xmmreg,imm	SSE41，
PEXTRB	Reg64，xmmreg，imm	X64
PEXTRD	rm32,xmmreg,imm	SSE41
PEXTRD	Rm32，xmmreg，imm	SSE41

		SSE41,X64
PEXTRQ	rm64,xmmreg,imm	SSE41，
PEXTRQ	Rm64，xmmreg，imm	X64
PEXTRW	reg32,xmmreg,imm	SSE41
PEXTRW	Reg32，xmmreg，imm	SSE41
PEXTRW	mem16,xmmreg,imm	SSE41
PEXTRW	Mem16，xmmreg，imm	SSE41
		SSE41,X64
PEXTRW	reg64,xmmreg,imm	SSE41，
PEXTRW	Reg64，xmmreg，imm	X64
PHMINPOSUW		
PHMINPOSUW	xmmreg,xmmrm	SSE41
PHMINPOSUW	Xmmreg，xmmrm	SSE41
PINSRB	xmmreg,mem,imm	SSE41
PINSRB	Xmmreg，mem，imm	SSE41
PINSRB	xmmreg,rm8,imm	SSE41
PINSRB	Xmmreg，rm8，imm	SSE41
PINSRB	xmmreg,reg32,imm	SSE41
PINSRB	Xmmreg，reg32，imm	SSE41

172

PINSRD	xmmreg,mem,imm	SSE41
PINSRD	Xmmreg，mem，imm	SSE41
PINSRD	xmmreg,rm32,imm	SSE41
PINSRD	Xmmreg，rm32，imm	SSE41
		SSE41,X64
PINSRQ	xmmreg,mem,imm	SSE41，
PINSRQ	Xmmreg，mem，imm	X64
		SSE41,X64
PINSRQ	xmmreg,rm64,imm	SSE41，
PINSRQ	Xmmreg，rm64，imm	X64
PMAXSB	xmmreg,xmmrm	SSE41
PMAXSB	Xmmreg，xmmrm	SSE41
PMAXSD	xmmreg,xmmrm	SSE41
PMAXSD	Xmmreg，xmmrm	SSE41
PMAXUD	xmmreg,xmmrm	SSE41
PMAXUD	Xmmreg，xmmrm	SSE41
PMAXUW	xmmreg,xmmrm	SSE41
PMAXUW	Xmmreg，xmmrm	SSE41
PMINSB	xmmreg,xmmrm	SSE41
PMINSB	Xmmreg，xmmrm	SSE41
PMINSD	xmmreg,xmmrm	SSE41
PMINSD	Xmmreg，xmmrm	SSE41
PMINUD	xmmreg,xmmrm	SSE41
PMINUD	Xmmreg，xmmrm	SSE41
PMINUW	xmmreg,xmmrm	SSE41
PMINUW	Xmmreg，xmmrm	SSE41
PMOVSXBW	xmmreg,xmmrm	SSE41
PMOVSXBW	Xmmreg，xmmrm	SSE41
PMOVSXBD	xmmreg,xmmrm	SSE41,SD
PMOVSXBD	Xmmreg，xmmrm	SSE41，SD
PMOVSXBQ	xmmreg,xmmrm	SSE41,SW
PMOVSXBQ	Xmmreg，xmmrm	SSE41，SW
PMOVSXWD	xmmreg,xmmrm	SSE41
PMOVSXWD	Xmmreg，xmmrm	SSE41
PMOVSXWQ	xmmreg,xmmrm	SSE41,SD
PMOVSXWQ	Xmmreg，xmmrm	SSE41，SD
PMOVSXDQ	xmmreg,xmmrm	SSE41
PMOVSXDQ	Xmmreg，xmmrm	SSE41
PMOVZXBW	xmmreg,xmmrm	SSE41
PMOVZXBW	Xmmreg，xmmrm	SSE41
PMOVZXBD	xmmreg,xmmrm	SSE41,SD
PMOVZXBD	Xmmreg，xmmrm	SSE41，SD
PMOVZXBQ	xmmreg,xmmrm	SSE41,SW
PMOVZXBQ	Xmmreg，xmmrm	SSE41，SW
PMOVZXWD	xmmreg,xmmrm	SSE41
PMOVZXWD	Xmmreg，xmmrm	SSE41
PMOVZXWQ	xmmreg,xmmrm	SSE41,SD
PMOVZXWQ	Xmmreg，xmmrm	SSE41，SD
PMOVZXDQ	xmmreg,xmmrm	SSE41
PMOVZXDQ	Xmmreg，xmmrm	SSE41

PMULDQ	xmmreg,xmmrm	SSE41
PMULDQ	Xmmreg，xmmrm	SSE41
PMULLD	xmmreg,xmmrm	SSE41
PMULLD	Xmmreg，xmmrm	SSE41
PTEST	xmmreg,xmmrm	SSE41
PTEST	Xmmreg，xmmrm	SSE41
ROUNDPD		
ROUNDPD 圆形	xmmreg,xmmrm,imm	SSE41
警察局	Xmmrg，xmmrm，imm	SSE41
ROUNDPS	xmmreg,xmmrm,imm	SSE41
圆规	Xmmrg，xmmrm，imm	SSE41
ROUNDSD		
ROUNDSD 圆桌	xmmreg,xmmrm,imm	SSE41
会议	Xmmrg，xmmrm，imm	SSE41
ROUNDSS	xmmreg,xmmrm,imm	SSE41
圆的	Xmmrg，xmmrm，imm	SSE41

B.1.19 Nehalem New Instructions (SSE4.2)
B. 1.19 Nehalem 新指令(SSE4.2)

CRC32	reg32,rm8	SSE42
CRC32	Reg32，rm8	SSE42
CRC32	reg32,rm16	SSE42
CRC32	Reg32，rm16	SSE42
CRC32	reg32,rm32	SSE42
CRC32	Reg32，rm32	SSE42
CRC32	reg64,rm8	SSE42,X64
CRC32	Reg64，rm8	SSE42 X64
CRC32	reg64,rm64	SSE42,X64
CRC32	Reg64，rm64	SSE42 X64
PCMPESTRI	xmmreg,xmmrm,imm	SSE42
PCMPESTRI	Xmmrg，xmmrm，imm	SSE42
PCMPESTRM	xmmreg,xmmrm,imm	SSE42
PCMPESTRM	Xmmrg，xmmrm，imm	SSE42
PCMPISTRI	xmmreg,xmmrm,imm	SSE42
PCMPISTRI	Xmmrg，xmmrm，imm	SSE42
PCMPISTRM	xmmreg,xmmrm,imm	SSE42
PCMPISTRM	Xmmrg，xmmrm，imm	SSE42
PCMPGTQ	xmmreg,xmmrm	SSE42
PCMPGTQ	Xmmreg，xmmrm	SSE42
POPCNT	reg16,rm16	NEHALEM,SW
POPCNT	正版 16，rm16	NEHALEM，SW
POPCNT	reg32,rm32	NEHALEM,SD
POPCNT	Reg32，rm32	NEHALEM，SD
POPCNT	reg64,rm64	NEHALEM,X64
POPCNT	Reg64，rm64	NEHALEM X64

B.1.20 Intel SMX
B. 1.20 英特尔 SMX

```
GETSEC                                    KATMAI
```
GETSEC KATMAI

B.1.21 Geode (Cyrix) 3DNow! additions
B. 1.21 Geode (Cyrix)3DNow

```
PFRCPV           mmxreg,mmxrm             PENT,3DNOW,CYRIX
```
PFRCPV mmxreg, mmxrm PENT, 3 DNOW, CYRIX

```
PFRSQRTV         mmxreg,mmxrm             PENT,3DNOW,CYRIX
```
PFRSQRTV mmxreg, mmxrm PENT, 3 DNOW, CYRIX

173

B.1.22 Intel new instructions in ???
B. 1.22 英特尔新指令？

MOVBE	reg16,mem16	NEHALEM
移动	Reg16，mem16	NEHALEM
MOVBE	reg32,mem32	NEHALEM
移动	Reg32，mem32	NEHALEM
MOVBE	reg64,mem64	NEHALEM
移动	Reg64，mem64	NEHALEM
MOVBE	mem16,reg16	NEHALEM
移动	Mem16，reg16	NEHALEM
MOVBE	mem32,reg32	NEHALEM
移动	Mem32，reg32	NEHALEM
MOVBE	mem64,reg64	NEHALEM
移动	Mem64，reg64	NEHALEM

B.1.23 Intel AES instructions
B. 1.23 英特尔 AES 指令

AESENC	xmmreg,xmmrm128	SSE,WESTMERE
AESENC	Xmmreg，xmmrm128	SSE，WESTMERE 西米尔
AESENCLAST	xmmreg,xmmrm128	SSE,WESTMERE
埃森哲	Xmmreg，xmmrm128	SSE，WESTMERE 西米尔
AESDEC	xmmreg,xmmrm128	SSE,WESTMERE
埃斯德克	Xmmreg，xmmrm128	SSE，WESTMERE 西米尔
AESDECLAST	xmmreg,xmmrm128	SSE,WESTMERE
AESDECLAST	Xmmreg，xmmrm128	SSE，WESTMERE 西米尔
AESIMC	xmmreg,xmmrm128	SSE,WESTMERE
AESIMC	Xmmreg，xmmrm128	SSE，WESTMERE 西米尔
AESKEYGENASSIST	xmmreg,xmmrm128,imm8	SSE,WESTMERE

AESKEYGENASSIST	Xmmreg，xmmmrm128，	SSE，
	imm8	WESTMERE 西
		米尔

B.1.24 Intel AVX AES instructions
英特尔 AVX AES 指令

	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VAESENC	Xmmreg，xmmreg *，	AVX，
VAESENC	xmmrm128	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VAESENCLAST	Xmmreg，xmmreg *，	AVX，
瓦森克拉斯特	xmmrm128	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VAESDEC	Xmmreg，xmmreg *，	AVX，
VAESDEC	xmmrm128	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VAESDECLAST	Xmmreg，xmmreg *，	AVX，
VAESDECLAST	xmmrm128	SANDYBRIDGE
		AVX,SANDYBRIDGE
VAESIMC	xmmreg,xmmrm128	AVX，
VAESIMC	Xmmreg，xmmrm128	SANDYBRIDGE
VAESKEYGENASSIST	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VAESKEYGENASSIST	Xmmreg，xmmmrm128，	AVX，
VAESKEYGENASSIST	imm8	SANDYBRIDGE

B.1.25 Intel instruction extension based on pub number 319433−030 dated October 2017
B. 1.25 英特尔指令扩展基于 2017 年 10 月发布号 319433-030

	ymmreg,ymmreg*,ymmrm256	
VAESENC	Ymmreg，ymmreg *，	VAES
VAESENC	ymmrm256	VAES
	ymmreg,ymmreg*,ymmrm256	
VAESENCLAST	Ymmreg，ymmreg *，	VAES
瓦森克拉斯特	ymmrm256	VAES
	ymmreg,ymmreg*,ymmrm256	
VAESDEC	Ymmreg，ymmreg *，	VAES
VAESDEC	ymmrm256	VAES
	ymmreg,ymmreg*,ymmrm256	
VAESDECLAST	Ymmreg，ymmreg *，	VAES
VAESDECLAST	ymmrm256	VAES
	xmmreg,xmmreg*,xmmrm128	AVX512VL,VAES
VAESENC	Xmmreg，xmmreg *，	AVX512VL，
VAESENC	xmmrm128	VAES
	ymmreg,ymmreg*,ymmrm256	AVX512VL,VAES
VAESENC	Ymmreg，ymmreg *，	AVX512VL，
VAESENC	ymmrm256	VAES
	xmmreg,xmmreg*,xmmrm128	AVX512VL,VAES
VAESENCLAST	Xmmreg，xmmreg *，	AVX512VL，
瓦森克拉斯特	xmmrm128	VAES
	ymmreg,ymmreg*,ymmrm256	AVX512VL,VAES
VAESENCLAST	Ymmreg，ymmreg *，	AVX512VL，
瓦森克拉斯特	ymmrm256	VAES

	xmmreg,xmmreg*,xmmrm128	AVX512VL,VAES
VAESDEC	Xmmreg，xmmreg * ,	AVX512VL，
VAESDEC	xmmrm128	VAES
	ymmreg,ymmreg*,ymmrm256	AVX512VL,VAES
VAESDEC	Ymmreg，ymmreg * ,	AVX512VL，
VAESDEC	ymmrm256	VAES
	xmmreg,xmmreg*,xmmrm128	AVX512VL,VAES
VAESDECLAST	Xmmreg，xmmreg * ,	AVX512VL，
VAESDECLAST	xmmrm128	VAES
	ymmreg,ymmreg*,ymmrm256	AVX512VL,VAES
VAESDECLAST	Ymmreg，ymmreg * ,	AVX512VL，
VAESDECLAST	ymmrm256	VAES
	zmmreg,zmmreg*,zmmrm512	
VAESENC	Zmmreg，zmmreg * ,	AVX512,VAES
VAESENC	zmmrm512	AVX512，VAES
	zmmreg,zmmreg*,zmmrm512	
VAESENCLAST	Zmmreg，zmmreg * ,	AVX512,VAES
瓦森克拉斯特	zmmrm512	AVX512，VAES
	zmmreg,zmmreg*,zmmrm512	
VAESDEC	Zmmreg，zmmreg * ,	AVX512,VAES
VAESDEC	zmmrm512	AVX512，VAES
	zmmreg,zmmreg*,zmmrm512	
VAESDECLAST	Zmmreg，zmmreg * ,	AVX512,VAES
VAESDECLAST	zmmrm512	AVX512，VAES

B.1.26 Intel AVX instructions
B. 1.26 英特尔 AVX 指令

	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VADDPD	Xmmreg，xmmreg * ,	AVX，
VADDPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VADDPD	Ymmreg，ymmreg * ,	AVX，
VADDPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VADDPS	Xmmreg，xmmreg * ,	AVX，
VADDPS	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VADDPS	Ymmreg，ymmreg * ,	AVX，
VADDPS	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VADDSD	Xmmreg，xmmreg * ,	AVX，
VADDSD	xmmrm64	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	AVX,SANDYBRIDGE
VADDSS	Xmmreg，xmmreg * ,	AVX，
VADDSS	xmmrm32	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VADDSUBPD	Xmmreg，xmmreg * ,	AVX，
VADDSUBPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VADDSUBPD	Ymmreg，ymmreg * ,	AVX，
VADDSUBPD	ymmrm256	SANDYBRIDGE

	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VADDSUBPS	Xmmreg， xmmreg * ，	AVX，
VADDSUBPS	xmmrm128	SANDYBRIDGE

174

	ymmreg,ymmreg*,ymmrm256	
VADDSUBPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VADDSUBPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VANDPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VANDPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VANDPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VANDPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VANDPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VANDPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VANDPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VANDPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VANDNPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VANDNPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VANDNPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VANDNPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VANDNPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VANDNPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VANDNPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VANDNPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VBLENDPD	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
VBLENDPD	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VBLENDPD	Ymmreg，ymmreg *，ymmrm256，imm8 AVX，	
VBLENDPD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VBLENDPS	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
VBLENDPS	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VBLENDPS	Ymmreg，ymmreg *，ymmrm256，imm8 AVX，	
VBLENDPS	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,xmmreg AVX,SANDYBRIDGE	
VBLENDVPD	Xmmreg，xmmreg *，xmmrm128，xmmreg AVX，	
VBLENDVPD	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,ymmreg AVX,SANDYBRIDGE	
VBLENDVPD	Ymmreg，ymmreg *，ymm256，ymmreg AVX，	
VBLENDVPD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,xmmreg AVX,SANDYBRIDGE	
VBLENDVPS	Xmmreg，xmmreg *，xmmrm128，xmmreg AVX，	
VBLENDVPS	SANDYBRIDGE	
VBLENDVPS	ymmreg,ymmreg*,ymmrm256,ymmreg AVX,SANDYBRIDGE	
VBLENDVPS	Ymmreg，ymmreg *，ymm256，ymmreg AVX，	

	SANDYBRIDGE	
VBROADCASTSS		
Vbroadcasts.vbroad	xmmreg,mem32	AVX,SANDYBRIDGE
cast	Xmmreg，mem32	AVX，SANDYBRIDGE
VBROADCASTSS		
Vbroadcasts.vbroad	ymmreg,mem32	AVX,SANDYBRIDGE
cast	Ymmreg，mem32	AVX，SANDYBRIDGE
VBROADCASTSD	ymmreg,mem64	AVX,SANDYBRIDGE
VBROADCASTSD	Ymmreg，mem64	AVX，SANDYBRIDGE
VBROADCASTF128	ymmreg,mem128	AVX,SANDYBRIDGE
VBROADCASTF128	Ymmreg mem128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPEQ_OSPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ ospd	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPEQ_OSPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ ospd	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPEQPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPEQPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPEQPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPEQPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPLT_OSPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPLT _ ospd	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLT_OSPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPLT _ ospd	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPLTPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPLTPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLTPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPLTPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPLE_OSPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
特种部队	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLE_OSPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
特种部队	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPLEPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPLEPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLEPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPLEPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPUNORD_QPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ qpd	xmmrm128	AVX，SANDYBRIDGE
VCMPUNORD_QPD	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE

VCMPUNORD _ qpd	Ymmreg，ymmreg *，	AVX，SANDYBRIDGE
	ymmrm256	
	xmmreg,xmmreg*,xmmrm128	
VCMPUNORDPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORDPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPUNORDPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPUNORDPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQ_UQPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ uqpd	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQ_UQPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ uqpd	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLT_USPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
美国警察	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLT_USPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
美国警察	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLTPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLTPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLTPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLTPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLE_USPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
美国警察	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLE_USPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
美国警察	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLEPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLEPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLEPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLEPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPORD_QPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORD _ qpd	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPORD_QPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPORD _ qpd	ymmrm256	AVX，SANDYBRIDGE
VCMPORDPD	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE

VCMPORDPD	Xmmreg， xmmreg *， xmmrm128 ymmreg,ymmreg*,ymmrm256	AVX， SANDYBRIDGE
VCMPORDPD	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPORDPD	ymmrm256 xmmreg,xmmreg*,xmmrm128	AVX， SANDYBRIDGE
VCMPEQ_UQPD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ uqpd	xmmrm128	AVX， SANDYBRIDGE

175

	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPEQ_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPEQ _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGE_USPD	Xmmreg，xmmreg *，	AVX，
VCMPNGE _ uspd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGE_USPD	Ymmreg，ymmreg *，	AVX，
VCMPNGE _ uspd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGEPD	Xmmreg，xmmreg *，	AVX，
VCMPNGEPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGEPD	Ymmreg，ymmreg *，	AVX，
VCMPNGEPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGT_USPD	Xmmreg，xmmreg *，	AVX，
VCMPNGT _ uspd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGT_USPD	Ymmreg，ymmreg *，	AVX，
VCMPNGT _ uspd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGTPD	Xmmreg，xmmreg *，	AVX，
VCMPNGTPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGTPD	Ymmreg，ymmreg *，	AVX，
VCMPNGTPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPFALSE_OQPD	Xmmreg，xmmreg *，	AVX，
VCMPFALSE _ oqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPFALSE_OQPD	Ymmreg，ymmreg *，	AVX，
VCMPFALSE _ oqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPFALSEPD	Xmmreg，xmmreg *，	AVX，
VCMPFALSEPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPFALSEPD	Ymmreg，ymmreg *，	AVX，
VCMPFALSEPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNEQ_OQPD	Xmmreg，xmmreg *，	AVX，
VCMPNEQ _ oqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNEQ_OQPD	Ymmreg，ymmreg *，	AVX，
VCMPNEQ _ oqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPGE_OSPD	Xmmreg，xmmreg *，	AVX，
奥斯佩德	xmmrm128	SANDYBRIDGE
VCMPGE_OSPD	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
奥斯佩德	Ymmreg，ymmreg *，	AVX，

	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPGEPD	Xmmreg，xmmreg *，	AVX，
VCMPGEPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPGEPD	Ymmreg，ymmreg *，	AVX，
VCMPGEPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPGT_OSPD	Xmmreg，xmmreg *，	AVX，
VCMPGT _ ospd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPGT_OSPD	Ymmreg，ymmreg *，	AVX，
VCMPGT _ ospd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPGTPD	Xmmreg，xmmreg *，	AVX，
VCMPGTPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPGTPD	Ymmreg，ymmreg *，	AVX，
VCMPGTPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPTRUE_UQPD	Xmmreg，xmmreg *，	AVX，
VCMPTRUE _ uqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPTRUE_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPTRUE _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPTRUEPD	Xmmreg，xmmreg *，	AVX，
VCMPTRUEPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPTRUEPD	Ymmreg，ymmreg *，	AVX，
VCMPTRUEPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPEQ_OSPD	Xmmreg，xmmreg *，	AVX，
VCMPEQ _ ospd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPEQ_OSPD	Ymmreg，ymmreg *，	AVX，
VCMPEQ _ ospd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPLT_OQPD	Xmmreg，xmmreg *，	AVX，
VCMPLT _ oqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPLT_OQPD	Ymmreg，ymmreg *，	AVX，
VCMPLT _ oqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPLE_OQPD	Xmmreg，xmmreg *，	AVX，
VCMPLE _ oqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPLE_OQPD	Ymmreg，ymmreg *，	AVX，
VCMPLE _ oqpd	ymmrm256	SANDYBRIDGE
VCMPUNORD_SPD	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPUNORD _ spd	Xmmreg，xmmreg *，	AVX，

	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPUNORD_SPD	Ymmreg，ymmreg *，	AVX，
VCMPUNORD _ spd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNEQ_USPD	Xmmreg，xmmreg *，	AVX，
VCMPNEQ _ uspd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNEQ_USPD	Ymmreg，ymmreg *，	AVX，
VCMPNEQ _ uspd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNLT_UQPD	Xmmreg，xmmreg *，	AVX，
VCMPNLT _ uqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNLT_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPNLT _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNLE_UQPD	Xmmreg，xmmreg *，	AVX，
VCMPNLE _ uqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNLE_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPNLE _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPORD_SPD	Xmmreg，xmmreg *，	AVX，
VCMPORD _ spd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPORD_SPD	Ymmreg，ymmreg *，	AVX，
VCMPORD _ spd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPEQ_USPD	Xmmreg，xmmreg *，	AVX，
VCMPEQ _ uspd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPEQ_USPD	Ymmreg，ymmreg *，	AVX，
VCMPEQ _ uspd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGE_UQPD	Xmmreg，xmmreg *，	AVX，
VCMPNGE _ uqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGE_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPNGE _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGT_UQPD	Xmmreg，xmmreg *，	AVX，
VCMPNGT _ uqpd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNGT_UQPD	Ymmreg，ymmreg *，	AVX，
VCMPNGT _ uqpd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPFALSE_OSPD	Xmmreg，xmmreg *，	AVX，
VCMPFALSE _ ospd	xmmrm128	SANDYBRIDGE
VCMPFALSE_OSPD	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPFALSE _ ospd	Ymmreg，ymmreg *，	AVX，

	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNEQ_OSPD	Xmmreg， xmmreg * ，	AVX，
VCMPNEQ _ ospd	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPNEQ_OSPD	Ymmreg， ymmreg * ，	AVX，
VCMPNEQ _ ospd	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPGE_OQPD	Xmmreg， xmmreg * ，	AVX，
VCMPGE _ oqpd	xmmrm128	SANDYBRIDGE

176

ymmreg,ymmreg*,ymmrm256
VCMPGE_OQPD Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPGE _ oqpd ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPGT_OQPD Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPGT _ oqpd xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPGT_OQPD Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPGT _ oqpd ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPTRUE_USPD Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPTRUE _ uspd xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPTRUE_USPD Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPTRUE _ uspd ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE
VCMPPD Xmmreg， xmmreg * ， xmmrm128， imm8 AVX，
VCMPPD SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE
VCMPPD Ymmreg， ymmreg * ， ymmrm256， imm8 AVX，
VCMPPD SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPEQ_OSPS Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPEQ _ osps xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPEQ_OSPS Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPEQ _ osps ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPEQPS Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPEQPS xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPEQPS Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPEQPS ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPLT_OSPS Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPLT _ osps xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPLT_OSPS Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPLT _ osps ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPLTPS Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
VCMPLTPS xmmrm128 AVX， SANDYBRIDGE
ymmreg,ymmreg*,ymmrm256
VCMPLTPS Ymmreg， ymmreg * ， AVX,SANDYBRIDGE
VCMPLTPS ymmrm256 AVX， SANDYBRIDGE
xmmreg,xmmreg*,xmmrm128
VCMPLE_OSPS Xmmreg， xmmreg * ， AVX,SANDYBRIDGE
Vcmple_osps xmmrm128 AVX， SANDYBRIDGE
VCMPLE_OSPS ymmreg,ymmreg*,ymmrm256 AVX,SANDYBRIDGE
Vcmple_osps Ymmreg， ymmreg * ， AVX， SANDYBRIDGE

	ymmrm256	
	xmmreg,xmmreg*,xmmrm128	
VCMPLEPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPLEPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLEPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPLEPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPUNORD_QPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ qps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPUNORD_QPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ qps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPUNORDPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORDPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPUNORDPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPUNORDPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQ_UQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ uqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQ_UQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ uqps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLT_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLT _ usps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLT_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLT _ usps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLTPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLTPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLTPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLTPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLE_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
美国邮政	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLE_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
美国邮政	ymmrm256	AVX，SANDYBRIDGE
VCMPNLEPS	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNLEPS	Xmmreg，xmmreg *，	AVX，SANDYBRIDGE

	xmmrm128	
	ymmreg,ymmreg*,ymmrm256	
VCMPNLEPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLEPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPORD_QPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORD _ qps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPORD_QPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPORD _ qps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPORDPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORDPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPORDPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPORDPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPEQ_UQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ uqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPEQ_UQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ uqps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNGE_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGE _ usps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNGE_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNGE _ usps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNGEPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGEPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNGEPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNGEPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNGT_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
美国邮政	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNGT_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
美国邮政	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNGTPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGTPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNGTPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNGTPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPFALSE_OQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPFALSE _ oqps	xmmrm128	AVX，SANDYBRIDGE
VCMPFALSE_OQPS	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPFALSE _ oqps	Ymmreg，ymmreg *，	AVX，SANDYBRIDGE

	ymmrm256	
	xmmreg,xmmreg*,xmmrm128	
VCMPFALSEPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPFALSEPS	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPFALSEPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPFALSEPS	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQ_OQPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNEQ _ oqps	xmmrm128	AVX， SANDYBRIDGE

177

	ymmreg,ymmreg*,ymmrm256	
VCMPNEQ_OQPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPNEQ _ oqps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGE_OSPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGE _ osps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGE_OSPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPGE _ osps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGEPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGEPS	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGEPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPGEPS	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGT_OSPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGT _ osps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGT_OSPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPGT _ osps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGTPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGTPS	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGTPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPGTPS	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPTRUE_UQPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPTRUE _ uqps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPTRUE_UQPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPTRUE _ uqps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPTRUEPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPTRUEPS	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPTRUEPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPTRUEPS	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPEQ_OSPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ osps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPEQ_OSPS	Ymmreg， ymmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ osps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPLT_OQPS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPLT _ oqps	xmmrm128	AVX， SANDYBRIDGE
VCMPLT_OQPS	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VCMPLT _ oqps	Ymmreg， ymmreg * ，	AVX， SANDYBRIDGE

	ymmrm256	
	xmmreg,xmmreg*,xmmrm128	
VCMPLE_OQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPLE _ oqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPLE_OQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPLE _ oqps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPUNORD_SPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ sps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPUNORD_SPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ sps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQ_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ usps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQ_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ usps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLT_UQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLT _ uqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLT_UQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLT _ uqps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNLE_UQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNLE _ uqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNLE_UQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNLE _ uqps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPORD_SPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORD _ sps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPORD_SPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPORD _ sps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPEQ_USPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ usps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPEQ_USPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ usps	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNGE_UQPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGE _ uqps	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNGE_UQPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VCMPNGE _ uqps	ymmrm256	AVX，SANDYBRIDGE
VCMPNGT_UQPS	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VCMPNGT _ uqps	Xmmreg，xmmreg *，	AVX，SANDYBRIDGE

	xmmrm128	
	ymmreg,ymmreg*,ymmrm256	
VCMPNGT_UQPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPNGT _ uqps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPFALSE_OSPS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPFALSE _ osps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPFALSE_OSPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPFALSE _ osps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPNEQ_OSPS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ osps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPNEQ_OSPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ osps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGE_OQPS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPGE _ oqps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGE_OQPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPGE _ oqps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPGT_OQPS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPGT _ oqps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPGT_OQPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPGT _ oqps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VCMPTRUE_USPS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPTRUE _ usps	xmmrm128	AVX， SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VCMPTRUE_USPS	Ymmreg， ymmreg *，	AVX,SANDYBRIDGE
VCMPTRUE _ usps	ymmrm256	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VCMPPS	Xmmreg， xmmreg *， xmmrm128， imm8 AVX，	
VCMPPS	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VCMPPS	Ymmreg， ymmreg *， ymmrm256， imm8 AVX，	
VCMPPS	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_OSSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPEQSD	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLT_OSSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPLT _ ossd	xmmrm64	AVX， SANDYBRIDGE
VCMPLTSD	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VCMPLTSD	Xmmreg， xmmreg *，	AVX， SANDYBRIDGE

	xmmrm64	
	xmmreg,xmmreg*,xmmrm64	
VCMPLE_OSSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
Vcmple_ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLESD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPLESD	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPUNORD_QSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPUNORD _ qsd	xmmrm64	AVX， SANDYBRIDGE

178

	xmmreg,xmmreg*,xmmrm64	
VCMPUNORDSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPUNORDSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_UQSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNEQ _ uqsd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNEQSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLT_USSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLT _ ussd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLTSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLTSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLE_USSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLE _ ussd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLESD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLESD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPORD_QSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPORD _ qsd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPORDSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPORDSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_UQSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ uqsd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGE_USSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNGE _ ussd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGESD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNGESD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGT_USSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
美国标准协会	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGTSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPNGTSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSE_OQSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPFALSE _ oqsd	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSESD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCMPFALSESD	xmmrm64	AVX，SANDYBRIDGE
VCMPNEQ_OQSD	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VCMPNEQ _ oqsd	Xmmreg，xmmreg * ，	AVX，SANDYBRIDGE

	xmmrm64	
	xmmreg,xmmreg*,xmmrm64	
VCMPGE_OSSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGE _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGESD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGESD	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGT_OSSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGT _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGTSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGTSD	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUE_UQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPTRUE _ uqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUESD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPTRUESD	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_OSSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLT_OQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPLT _ oqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLE_OQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPLE _ oqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPUNORD_SSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPUNORD _ ssd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_USSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNEQ _ ussd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLT_UQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLT _ uqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLE_UQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLE _ uqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPORD_SSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPORD _ ssd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_USSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ ussd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGE_UQSD	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
Vcmpnge_uqsd	xmmrm64	AVX， SANDYBRIDGE
VCMPNGT_UQSD	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VCMPNGT _ uqsd	Xmmreg， xmmreg * ，	AVX， SANDYBRIDGE

	xmmrm64	
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSE_OSSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPFALSE _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_OSSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ ossd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGE_OQSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
Vcmpge_oqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGT_OQSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPGT _ oqsd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUE_USSD	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPTRUE _ ussd	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64,imm8 AVX,SANDYBRIDGE	
VCMPSD	Xmmreg， xmmreg *， xmmrm64， imm8 AVX，	
VCMPSD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_OSSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ osss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPEQSS	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLT_OSSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPLT _ osss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLTSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPLTSS	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLE_OSSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPLE _ osss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLESS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPLESS	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPUNORD_QSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORD _ qss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPUNORDSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPUNORDSS	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_UQSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ uqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQSS	Xmmreg， xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQSS	xmmrm64	AVX， SANDYBRIDGE
VCMPNLT_USSS	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VCMPNLT	Xmmreg， xmmreg *，	AVX， SANDYBRIDGE

	xmmrm64	
	xmmreg,xmmreg*,xmmrm64	
VCMPNLTSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLTSS	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLE_USSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
美国海军	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLESS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLESS	xmmrm64	AVX， SANDYBRIDGE

179

	xmmreg,xmmreg*,xmmrm64	
VCMPORD_QSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORD _ qss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPORDSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPORDSS	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_UQSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPEQ _ uqss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGE_USSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGE _ uss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGESS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
Vcmpgess	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGT_USSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGT	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGTSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNGTSS	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSE_OQSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPFALSE _ oqss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSESS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
Vcmpfaless	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_OQSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPNEQ _ oqss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGE_OSSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
Vcmpge_osss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGESS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPGESS	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGT_OSSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPGT _ osss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGTSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPGTSS	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUE_UQSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPTRUE _ uqss	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUESS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCMPTRUESS	xmmrm64	AVX，SANDYBRIDGE
VCMPEQ_OSSS	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VCMPEQ _ osss	Xmmreg，xmmreg *，	AVX，SANDYBRIDGE

	xmmrm64	
	xmmreg,xmmreg*,xmmrm64	
VCMPLT_OQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPLT _ oqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPLE_OQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
Vcmple_oqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPUNORD_SSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPUNORD _ sss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_USSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
美国海军陆战队	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLT_UQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLT _ uqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNLE_UQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNLE _ uqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPORD_SSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPORD _ sss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPEQ_USSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPEQ _ usss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGE_UQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNGE _ uqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNGT_UQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNGT _ uqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPFALSE_OSSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPFALSE _ osss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPNEQ_OSSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPNEQ _ osss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGE_OQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGE _ oqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPGT_OQSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPGT _ oqss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VCMPTRUE_USSS	Xmmreg， xmmreg * ，	AVX,SANDYBRIDGE
VCMPTRUE _ usss	xmmrm64	AVX， SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64,imm8	AVX,SANDYBRIDGE
VCMPSS	Xmmreg， xmmreg * ， xmmrm64， imm8 AVX，	
VCMPSS	SANDYBRIDGE	
VCOMISD	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VCOMISD	Xmmreg， xmmrm64	AVX， SANDYBRIDGE

VCOMISS	xmmreg,xmmrm32	AVX,SANDYBRIDGE
VCOMISS	Xmmrm32	AVX，SANDYBRIDGE
VCVTDQ2PD	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VCVTDQ2PD	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VCVTDQ2PD	ymmreg,xmmrm128	AVX,SANDYBRIDGE
VCVTDQ2PD	Ymmreg，xmmrm128	AVX，SANDYBRIDGE
VCVTDQ2PS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VCVTDQ2PS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VCVTDQ2PS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VCVTDQ2PS	Ymmreg，ymm256	AVX，SANDYBRIDGE
VCVTPD2DQ	xmmreg,xmmreg	AVX,SANDYBRIDGE
VCVTPD2DQ	Xmmreg，xmmreg	AVX，SANDYBRIDGE
VCVTPD2DQ	xmmreg,mem128	AVX,SANDYBRIDGE,SO
VCVTPD2DQ	Xmmreg，mem128	AVX，SANDYBRIDGE，SO
VCVTPD2DQ	xmmreg,ymmreg	AVX,SANDYBRIDGE
VCVTPD2DQ	Xmreg，ymmreg	AVX，SANDYBRIDGE
VCVTPD2DQ	xmmreg,mem256	AVX,SANDYBRIDGE,SY
VCVTPD2DQ	Xmmreg，mem256	AVX，SANDYBRIDGE，SY
VCVTPD2PS	xmmreg,xmmreg	AVX,SANDYBRIDGE
VCVTPD2PS	Xmmreg，xmmreg	AVX，SANDYBRIDGE
VCVTPD2PS	xmmreg,mem128	AVX,SANDYBRIDGE,SO
VCVTPD2PS	Xmmreg，mem128	AVX，SANDYBRIDGE，SO
VCVTPD2PS	xmmreg,ymmreg	AVX,SANDYBRIDGE
VCVTPD2PS	Xmreg，ymmreg	AVX，SANDYBRIDGE
VCVTPD2PS	xmmreg,mem256	AVX,SANDYBRIDGE,SY
VCVTPD2PS	Xmmreg，mem256	AVX，SANDYBRIDGE，SY
VCVTPS2DQ	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VCVTPS2DQ	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VCVTPS2DQ	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VCVTPS2DQ	Ymmreg，ymm256	AVX，SANDYBRIDGE
VCVTPS2PD	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VCVTPS2PD	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VCVTPS2PD	ymmreg,xmmrm128	AVX,SANDYBRIDGE
VCVTPS2PD	Ymmreg，xmmrm128	AVX，SANDYBRIDGE
VCVTSD2SI	reg32,xmmrm64	AVX,SANDYBRIDGE
VCVTSD2SI	Reg32，xmmrm64	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VCVTSD2SI	reg64,xmmrm64	AVX，SANDYBRIDGE，
VCVTSD2SI	Reg64，xmmrm64	LONG
	xmmreg,xmmreg*,xmmrm64	
VCVTSD2SS	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VCVTSD2SS	xmmrm64	AVX，SANDYBRIDGE

180

VCVTSI2SD	xmmreg,xmmreg*,rm32	AVX,SANDYBRIDGE,SD
VCVTSI2SD	Xmmreg，xmmreg *， rm32	AVX，SANDYBRIDGE，SD
		AVX,SANDYBRIDGE,ND,SD
VCVTSI2SD	xmmreg,xmmreg*,mem32	AVX，SANDYBRIDGE，
VCVTSI2SD	Xmmreg，xmmreg *， mem32	ND，SD
		AVX,SANDYBRIDGE,LONG
VCVTSI2SD	xmmreg,xmmreg*,rm64	AVX，SANDYBRIDGE，
VCVTSI2SD	Xmmreg，xmmreg *， rm64	LONG
VCVTSI2SS		
VCVTSI2SS	xmmreg,xmmreg*,rm32	AVX,SANDYBRIDGE,SD
VCVTSI2SS	Xmmreg，xmmreg *， rm32	AVX，SANDYBRIDGE，SD
VCVTSI2SS		AVX,SANDYBRIDGE,ND,SD
VCVTSI2SS	xmmreg,xmmreg*,mem32	AVX，SANDYBRIDGE，
VCVTSI2SS	Xmmreg，xmmreg *， mem32	ND，SD
VCVTSI2SS		AVX,SANDYBRIDGE,LONG
VCVTSI2SS	xmmreg,xmmreg*,rm64	AVX，SANDYBRIDGE，
VCVTSI2SS	Xmmreg，xmmreg *， rm64	LONG
	xmmreg,xmmreg*,xmmrm32	
VCVTSS2SD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VCVTSS2SD	xmmrm32	AVX，SANDYBRIDGE
VCVTSS2SI	reg32,xmmrm32	AVX,SANDYBRIDGE
VCVTSS2SI	Reg32，xmmrm32	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VCVTSS2SI	reg64,xmmrm32	AVX，SANDYBRIDGE，
VCVTSS2SI	Reg64，xmmrm32	LONG
VCVTTPD2DQ	xmmreg,xmmreg	AVX,SANDYBRIDGE
VCVTTPD2DQ	Xmmreg，xmmreg	AVX，SANDYBRIDGE
VCVTTPD2DQ	xmmreg,mem128	AVX,SANDYBRIDGE,SO
VCVTTPD2DQ	Xmmreg，mem128	AVX，SANDYBRIDGE，SO
VCVTTPD2DQ	xmmreg,ymmreg	AVX,SANDYBRIDGE
VCVTTPD2DQ	Xmreg，ymmreg	AVX，SANDYBRIDGE
VCVTTPD2DQ	xmmreg,mem256	AVX,SANDYBRIDGE,SY
VCVTTPD2DQ	Xmmreg，mem256	AVX，SANDYBRIDGE，SY
VCVTTPS2DQ	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VCVTTPS2DQ	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VCVTTPS2DQ	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VCVTTPS2DQ	Ymmreg，ymm256	AVX，SANDYBRIDGE
VCVTTSD2SI	reg32,xmmrm64	AVX,SANDYBRIDGE
VCVTTSD2SI	Reg32，xmmrm64	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VCVTTSD2SI	reg64,xmmrm64	AVX，SANDYBRIDGE，
VCVTTSD2SI	Reg64，xmmrm64	LONG
VCVTTSS2SI	reg32,xmmrm32	AVX,SANDYBRIDGE
VCVTTSS2SI	Reg32，xmmrm32	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VCVTTSS2SI	reg64,xmmrm32	AVX，SANDYBRIDGE，
VCVTTSS2SI	Reg64，xmmrm32	LONG
	xmmreg,xmmreg*,xmmrm128	
VDIVPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VDIVPD	xmmrm128	AVX，SANDYBRIDGE

	ymmreg,ymmreg*,ymmrm256	
VDIVPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VDIVPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VDIVPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VDIVPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VDIVPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VDIVPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VDIVSD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VDIVSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VDIVSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VDIVSS	xmmrm32	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VDPPD	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
VDPPD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VDPPS	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
VDPPS	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VDPPS	Ymmreg，ymmreg *，ymmrm256，imm8 AVX，	
VDPPS	SANDYBRIDGE	
VEXTRACTF128	xmmrm128,ymmreg,imm8	AVX,SANDYBRIDGE
VEXTRACTF128	Xmmrm128，ymmreg，imm8	AVX，SANDYBRIDGE
VEXTRACTPS	rm32,xmmreg,imm8	AVX,SANDYBRIDGE
VEXTRACTPS	Rm32，xmmreg，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VHADDPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VHADDPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VHADDPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VHADDPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VHADDPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VHADDPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VHADDPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VHADDPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VHSUBPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VHSUBPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VHSUBPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VHSUBPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VHSUBPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VHSUBPS	xmmrm128	AVX，SANDYBRIDGE
VHSUBPS	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VHSUBPS	Ymmreg，ymmreg *，	AVX，SANDYBRIDGE

	ymmrm256	
	ymmreg,ymmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VINSERTF128	Ymmreg，ymmreg *，xmmrm128，imm8 AVX，	
VINSERTF128	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm32,imm8 AVX,SANDYBRIDGE	
VINSERTPS	Xmmreg，xmmreg *，xmmrm32，imm8 AVX，	
VINSERTPS	SANDYBRIDGE	
VLDDQU	xmmreg,mem128	AVX,SANDYBRIDGE
VLDDQU	Xmmreg，mem128	AVX，SANDYBRIDGE
VLDQQU	ymmreg,mem256	AVX,SANDYBRIDGE
VLDQQU	Ymmreg，mem256	AVX，SANDYBRIDGE
VLDDQU	ymmreg,mem256	AVX,SANDYBRIDGE
VLDDQU	Ymmreg，mem256	AVX，SANDYBRIDGE
VLDMXCSR	mem32	AVX,SANDYBRIDGE
VLDMXCSR	Mem32	AVX，SANDYBRIDGE
VMASKMOVDQU	xmmreg,xmmreg	AVX,SANDYBRIDGE
VMASKMOVDQU	Xmmreg，xmmreg	AVX，SANDYBRIDGE
VMASKMOVPS		
VMASKMOVPS	xmmreg,xmmreg,mem128	AVX,SANDYBRIDGE
VMASKMOVPS	Xmmreg，xmmreg，mem128	AVX，SANDYBRIDGE
VMASKMOVPS		
VMASKMOVPS	ymmreg,ymmreg,mem256	AVX,SANDYBRIDGE
VMASKMOVPS	Ymmreg，ymmreg，mem256	AVX，SANDYBRIDGE
VMASKMOVPS		
VMASKMOVPS	mem128,xmmreg,xmmreg	AVX,SANDYBRIDGE,SO
VMASKMOVPS	Mem128，xmmreg，xmmreg	AVX，SANDYBRIDGE，SO
VMASKMOVPS		
VMASKMOVPS	mem256,ymmreg,ymmreg	AVX,SANDYBRIDGE,SY
VMASKMOVPS	Mem256 ymmreg ymmreg	AVX，SANDYBRIDGE，SY
VMASKMOVPD	xmmreg,xmmreg,mem128	AVX,SANDYBRIDGE
VMASKMOVPD	Xmmreg，xmmreg，mem128	AVX，SANDYBRIDGE
VMASKMOVPD	ymmreg,ymmreg,mem256	AVX,SANDYBRIDGE
VMASKMOVPD	Ymmreg，ymmreg，mem256	AVX，SANDYBRIDGE
VMASKMOVPD	mem128,xmmreg,xmmreg	AVX,SANDYBRIDGE
VMASKMOVPD	Mem128，xmmreg，xmmreg	AVX，SANDYBRIDGE
VMASKMOVPD	mem256,ymmreg,ymmreg	AVX,SANDYBRIDGE
VMASKMOVPD	Mem256 ymmreg ymmreg	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VMAXPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMAXPD	xmmrm128	AVX，SANDYBRIDGE

	ymmreg,ymmreg*,ymmrm256	
VMAXPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VMAXPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VMAXPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMAXPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VMAXPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VMAXPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VMAXSD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMAXSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VMAXSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMAXSS	xmmrm32	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VMINPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMINPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VMINPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VMINPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VMINPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMINPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VMINPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VMINPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VMINSD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMINSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VMINSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VMINSS	xmmrm32	AVX，SANDYBRIDGE
VMOVAPD	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVAPD	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVAPD	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVAPD	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVAPD	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVAPD	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVAPD	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVAPD	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVAPS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVAPS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVAPS	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVAPS	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVAPS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVAPS	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVAPS	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVAPS	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVD	xmmreg,rm32	AVX,SANDYBRIDGE

VMOVD	Xmmreg，rm32	AVX，SANDYBRIDGE
VMOVD	rm32,xmmreg	AVX,SANDYBRIDGE
VMOVD	Rm32，xmmreg	AVX，SANDYBRIDGE
VMOVQ	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VMOVQ	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VMOVQ	xmmrm64,xmmreg	AVX,SANDYBRIDGE
VMOVQ	Xmmrm64，xmreg	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VMOVQ	xmmreg,rm64	AVX，SANDYBRIDGE，
VMOVQ	Xmmreg，rm64	LONG
		AVX,SANDYBRIDGE,LONG
VMOVQ	rm64,xmmreg	AVX，SANDYBRIDGE，
VMOVQ	Rm64，xmmreg	LONG
VMOVDDUP	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VMOVDDUP	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VMOVDDUP	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVDDUP	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVDQA	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVDQA	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVDQA	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVDQA	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVQQA	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVQQA	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVQQA	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVQQA	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVDQA	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVDQA	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVDQA	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVDQA	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVDQU	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVDQU	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVDQU	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVDQU	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVQQU	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVQQU	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVQQU	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVQQU	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVDQU	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVDQU	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVDQU	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVDQU	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVHLPS	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVHLPS	Xmmreg，xmmreg *，xmmreg	AVX，SANDYBRIDGE
VMOVHPD	xmmreg,xmmreg*,mem64	AVX,SANDYBRIDGE
VMOVHPD	Xmmreg，xmmreg *，mem64	AVX，SANDYBRIDGE
VMOVHPD	mem64,xmmreg	AVX,SANDYBRIDGE
VMOVHPD	Mem64，xmmreg	AVX，SANDYBRIDGE
VMOVHPS	xmmreg,xmmreg*,mem64	AVX,SANDYBRIDGE
VMOVHPS	Xmmreg，xmmreg *，mem64	AVX，SANDYBRIDGE
VMOVHPS	mem64,xmmreg	AVX,SANDYBRIDGE

VMOVHPS	Mem64，xmmreg	AVX，SANDYBRIDGE
VMOVLHPS	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVLHPS	Xmmreg，xmmreg *， xmmreg	AVX，SANDYBRIDGE
VMOVLPD	xmmreg,xmmreg*,mem64	AVX,SANDYBRIDGE
VMOVLPD	Xmmreg，xmmreg *， mem64	AVX，SANDYBRIDGE
VMOVLPD	mem64,xmmreg	AVX,SANDYBRIDGE
VMOVLPD	Mem64，xmmreg	AVX，SANDYBRIDGE
VMOVLPS	xmmreg,xmmreg*,mem64	AVX,SANDYBRIDGE
VMOVLPS	Xmmreg，xmmreg *， mem64	AVX，SANDYBRIDGE
VMOVLPS	mem64,xmmreg	AVX,SANDYBRIDGE
VMOVLPS	Mem64，xmmreg	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VMOVMSKPD	reg64,xmmreg	AVX，SANDYBRIDGE，
VMOVMSKPD	Reg64，xmmreg	LONG
VMOVMSKPD	reg32,xmmreg	AVX,SANDYBRIDGE
VMOVMSKPD	Reg32，xmmreg	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VMOVMSKPD	reg64,ymmreg	AVX，SANDYBRIDGE，
VMOVMSKPD	Reg64，ymmreg	LONG
VMOVMSKPD	reg32,ymmreg	AVX,SANDYBRIDGE
VMOVMSKPD	32 岁，年轻人	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VMOVMSKPS	reg64,xmmreg	AVX，SANDYBRIDGE，
VMOVMSKPS	Reg64，xmmreg	LONG

182

VMOVMSKPS	reg32,xmmreg	AVX,SANDYBRIDGE
VMOVMSKPS	Reg32，xmmreg	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VMOVMSKPS	reg64,ymmreg	AVX，SANDYBRIDGE，
VMOVMSKPS	Reg64，ymmreg	LONG
VMOVMSKPS	reg32,ymmreg	AVX,SANDYBRIDGE
VMOVMSKPS	32 岁，年轻人	AVX，SANDYBRIDGE
VMOVNTDQ	mem128,xmmreg	AVX,SANDYBRIDGE
VMOVNTDQ	Mem128，xmmreg	AVX，SANDYBRIDGE
VMOVNTQQ	mem256,ymmreg	AVX,SANDYBRIDGE
VMOVNTQQ	Mem256，ymmreg	AVX，SANDYBRIDGE
VMOVNTDQ	mem256,ymmreg	AVX,SANDYBRIDGE
VMOVNTDQ	Mem256，ymmreg	AVX，SANDYBRIDGE
VMOVNTDQA	xmmreg,mem128	AVX,SANDYBRIDGE
VMOVNTDQA	Xmmreg，mem128	AVX，SANDYBRIDGE
VMOVNTPD	mem128,xmmreg	AVX,SANDYBRIDGE
VMOVNTPD	Mem128，xmmreg	AVX，SANDYBRIDGE
VMOVNTPD	mem256,ymmreg	AVX,SANDYBRIDGE
VMOVNTPD	Mem256，ymmreg	AVX，SANDYBRIDGE
VMOVNTPS	mem128,xmmreg	AVX,SANDYBRIDGE
VMOVNTPS	Mem128，xmmreg	AVX，SANDYBRIDGE
VMOVNTPS	mem256,ymmreg	AVX,SANDYBRIDGE
VMOVNTPS	Mem256，ymmreg	AVX，SANDYBRIDGE
VMOVSD	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVSD	Xmmreg，xmmreg *，xmmreg	AVX，SANDYBRIDGE
VMOVSD	xmmreg,mem64	AVX,SANDYBRIDGE
VMOVSD	Xmmreg，mem64	AVX，SANDYBRIDGE
VMOVSD	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVSD	Xmmreg，xmmreg *，xmmreg	AVX，SANDYBRIDGE
VMOVSD	mem64,xmmreg	AVX,SANDYBRIDGE
VMOVSD	Mem64，xmmreg	AVX，SANDYBRIDGE
VMOVSHDUP	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVSHDUP	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVSHDUP	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVSHDUP	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVSLDUP	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVSLDUP	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVSLDUP	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVSLDUP	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVSS		
VMOVSS	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVSS	Xmmreg，xmmreg *，xmmreg	AVX，SANDYBRIDGE
VMOVSS		
VMOVSS	xmmreg,mem32	AVX,SANDYBRIDGE
VMOVSS	Xmmreg，mem32	AVX，SANDYBRIDGE
VMOVSS		
VMOVSS	xmmreg,xmmreg*,xmmreg	AVX,SANDYBRIDGE
VMOVSS	Xmmreg，xmmreg *，xmmreg	AVX，SANDYBRIDGE
VMOVSS	mem32,xmmreg	AVX,SANDYBRIDGE
VMOVSS	Mem32，xmmreg	AVX，SANDYBRIDGE

VMOVSS		
VMOVUPD	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVUPD	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVUPD	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVUPD	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVUPD	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVUPD	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVUPD	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVUPD	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMOVUPS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VMOVUPS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VMOVUPS	xmmrm128,xmmreg	AVX,SANDYBRIDGE
VMOVUPS	Xmmrm128，xmreg	AVX，SANDYBRIDGE
VMOVUPS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VMOVUPS	Ymmreg，ymm256	AVX，SANDYBRIDGE
VMOVUPS	ymmrm256,ymmreg	AVX,SANDYBRIDGE
VMOVUPS	Ymmrm256，ymmreg	AVX，SANDYBRIDGE
VMPSADBW	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VMPSADBW	Xmmreg，xmmreg＊，xmmrm128，imm8 AVX，	
VMPSADBW	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128	
VMULPD	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VMULPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VMULPD	Ymmreg，ymmreg＊，	AVX,SANDYBRIDGE
VMULPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VMULPS	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VMULPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VMULPS	Ymmreg，ymmreg＊，	AVX,SANDYBRIDGE
VMULPS	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VMULSD	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VMULSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VMULSS	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VMULSS	xmmrm32	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VORPD	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VORPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VORPD	Ymmreg，ymmreg＊，	AVX,SANDYBRIDGE
VORPD	ymmrm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VORPS	Xmmreg，xmmreg＊，	AVX,SANDYBRIDGE
VORPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VORPS	Ymmreg，ymmreg＊，	AVX,SANDYBRIDGE
VORPS	ymmrm256	AVX，SANDYBRIDGE
VPABSB	xmmreg,xmmrm128	AVX,SANDYBRIDGE

VPABSB	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VPABSW	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VPABSW	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VPABSD	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VPABSD	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPACKSSWB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPACKSSWB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPACKSSDW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPACKSSDW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPACKUSWB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPACKUSWB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPACKUSDW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPACKUSDW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDSB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDSB	xmmrm128	AVX，SANDYBRIDGE

183

	xmmreg,xmmreg*,xmmrm128	
VPADDSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDUSB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDUSB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPADDUSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPADDUSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128,imm8	AVX,SANDYBRIDGE
VPALIGNR	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
VPALIGNR	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128	
VPAND	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPAND	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPANDN	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPANDN	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPAVGB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPAVGB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPAVGW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPAVGW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128,xmmreg	AVX,SANDYBRIDGE
VPBLENDVB	Xmmreg，xmmreg *，xmmrm128，xmmreg AVX，	
VPBLENDVB	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,imm8	AVX,SANDYBRIDGE
VPBLENDW	Xmmreg，xmmreg *，xmmrm128，imm8 AVX，	
翻译	SANDYBRIDGE	
VPCMPESTRI	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPCMPESTRI	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPCMPESTRM	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPCMPESTRM	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPCMPISTRI	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPCMPISTRI	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPCMPISTRM	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPCMPISTRM	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPEQB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPEQB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPEQW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPEQW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPEQD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPEQD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPEQQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPEQQ	xmmrm128	AVX，SANDYBRIDGE

	xmmreg,xmmreg*,xmmrm128	
VPCMPGTB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPGTB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPGTW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPGTW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPGTD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPGTD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPCMPGTQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPCMPGTQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPERMILPD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPERMILPD	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VPERMILPD	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VPERMILPD	ymmrm256	AVX，SANDYBRIDGE
VPERMILPD	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPERMILPD	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPERMILPD	ymmreg,ymmrm256,imm8	AVX,SANDYBRIDGE
VPERMILPD	Ymmreg，ymmrm256，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPERMILPS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPERMILPS	xmmrm128	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	
VPERMILPS	Ymmreg，ymmreg *，	AVX,SANDYBRIDGE
VPERMILPS	ymmrm256	AVX，SANDYBRIDGE
VPERMILPS	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPERMILPS	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPERMILPS	ymmreg,ymmrm256,imm8	AVX,SANDYBRIDGE
VPERMILPS	Ymmreg，ymmrm256，imm8	AVX，SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256,imm8	AVX,SANDYBRIDGE
VPERM2F128	Ymmreg，ymmreg *， ymmrm256，imm8 AVX，	
VPERM2F128	SANDYBRIDGE	
		AVX,SANDYBRIDGE,LONG
VPEXTRB	reg64,xmmreg,imm8	AVX，SANDYBRIDGE，
VPEXTRB	Reg64，xmmreg，imm8	LONG
VPEXTRB	reg32,xmmreg,imm8	AVX,SANDYBRIDGE
VPEXTRB	Reg32，xmmreg，imm8	AVX，SANDYBRIDGE
VPEXTRB	mem8,xmmreg,imm8	AVX,SANDYBRIDGE
VPEXTRB	Mem8，xmmreg，imm8	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VPEXTRW	reg64,xmmreg,imm8	AVX，SANDYBRIDGE，
VPEXTRW	Reg64，xmmreg，imm8	LONG
VPEXTRW	reg32,xmmreg,imm8	AVX,SANDYBRIDGE
VPEXTRW	Reg32，xmmreg，imm8	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VPEXTRW	reg64,xmmreg,imm8	AVX，SANDYBRIDGE，
VPEXTRW	Reg64，xmmreg，imm8	LONG
VPEXTRW	reg32,xmmreg,imm8	AVX,SANDYBRIDGE

VPEXTRW	Reg32，xmmreg，imm8	AVX，SANDYBRIDGE
VPEXTRW	mem16,xmmreg,imm8	AVX,SANDYBRIDGE
VPEXTRW	Mem16，xmmreg，imm8	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VPEXTRD	reg64,xmmreg,imm8	AVX，SANDYBRIDGE，
VPEXTRD	Reg64，xmmreg，imm8	LONG
VPEXTRD	rm32,xmmreg,imm8	AVX,SANDYBRIDGE
VPEXTRD	Rm32，xmmreg，imm8	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VPEXTRQ	rm64,xmmreg,imm8	AVX，SANDYBRIDGE，
VPEXTRQ	Rm64，xmmreg，imm8	LONG
	xmmreg,xmmreg*,xmmrm128	
VPHADDW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHADDW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPHADDD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHADDD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPHADDSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHADDSW	xmmrm128	AVX，SANDYBRIDGE
VPHMINPOSUW	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VPHMINPOSUW	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VPHSUBW	xmmreg,xmmreg*,xmmrm128	
VPHSUBW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHSUBW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPHSUBD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHSUBD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPHSUBSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPHSUBSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,mem8,imm8	
VPINSRB	Xmmreg，xmmreg *， mem8，	AVX,SANDYBRIDGE
VPINSRB	imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,rm8,imm8	
VPINSRB	Xmmreg，xmmreg *， rm8，	AVX,SANDYBRIDGE
VPINSRB	imm8	AVX，SANDYBRIDGE
VPINSRB	xmmreg,xmmreg*,reg32,imm8	AVX,SANDYBRIDGE
VPINSRB	Xmmreg，xmmreg *， reg32，imm8	AVX，SANDYBRIDGE
VPINSRW	xmmreg,xmmreg*,mem16,imm8	AVX,SANDYBRIDGE
VPINSRW	Xmmreg，xmmreg *， mem16，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,rm16,imm8	
VPINSRW	Xmmreg，xmmreg *， rm16，	AVX,SANDYBRIDGE
VPINSRW	imm8	AVX，SANDYBRIDGE

184

VPINSRW	xmmreg,xmmreg*,reg32,imm8	AVX,SANDYBRIDGE
VPINSRW	Xmmreg，xmmreg *，reg32，imm8	AVX，SANDYBRIDGE
VPINSRD	xmmreg,xmmreg*,mem32,imm8	AVX,SANDYBRIDGE
VPINSRD	Xmmreg，xmmreg *，mem32，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,rm32,imm8	
VPINSRD	Xmmreg，xmmreg *，rm32，	AVX,SANDYBRIDGE
VPINSRD	imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,mem64,imm8	AVX,SANDYBRIDGE,LONG
VPINSRQ	Xmmreg，xmmreg *，mem64，imm8	AVX，
VPINSRQ	SANDYBRIDGE，LONG	
	xmmreg,xmmreg*,rm64,imm8	AVX,SANDYBRIDGE,LONG
VPINSRQ	Xmmreg，xmmreg *，rm64，	AVX，SANDYBRIDGE，
VPINSRQ	imm8	LONG
	xmmreg,xmmreg*,xmmrm128	
VPMADDWD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMADDWD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMADDUBSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMADDUBSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXSB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXSB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXSD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXSD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXUB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXUB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXUW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXUW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMAXUD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMAXUD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMINSB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMINSB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMINSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMINSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMINSD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMINSD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMINUB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMINUB	xmmrm128	AVX，SANDYBRIDGE
VPMINUW	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE

VPMINUW	Xmmreg，xmmreg *，	AVX，SANDYBRIDGE
	xmmrm128	
	xmmreg,xmmreg*,xmmrm128	
VPMINUD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMINUD	xmmrm128	AVX，SANDYBRIDGE
		AVX,SANDYBRIDGE,LONG
VPMOVMSKB	reg64,xmmreg	AVX，SANDYBRIDGE，
VPMOVMSKB	Reg64，xmmreg	LONG
VPMOVMSKB	reg32,xmmreg	AVX,SANDYBRIDGE
VPMOVMSKB	Reg32，xmmreg	AVX，SANDYBRIDGE
VPMOVSXBW	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVSXBW	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VPMOVSXBD	xmmreg,xmmrm32	AVX,SANDYBRIDGE
VPMOVSXBD	Xmmrm32	AVX，SANDYBRIDGE
VPMOVSXBQ	xmmreg,xmmrm16	AVX,SANDYBRIDGE
VPMOVSXBQ	Xmmreg，xmmrm16	AVX，SANDYBRIDGE
VPMOVSXWD	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVSXWD	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VPMOVSXWQ	xmmreg,xmmrm32	AVX,SANDYBRIDGE
VPMOVSXWQ	Xmmrm32	AVX，SANDYBRIDGE
VPMOVSXDQ	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVSXDQ	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VPMOVZXBW	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVZXBW	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VPMOVZXBD	xmmreg,xmmrm32	AVX,SANDYBRIDGE
VPMOVZXBD	Xmmrm32	AVX，SANDYBRIDGE
VPMOVZXBQ	xmmreg,xmmrm16	AVX,SANDYBRIDGE
VPMOVZXBQ	Xmmreg，xmmrm16	AVX，SANDYBRIDGE
VPMOVZXWD	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVZXWD	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
VPMOVZXWQ	xmmreg,xmmrm32	AVX,SANDYBRIDGE
VPMOVZXWQ	Xmmrm32	AVX，SANDYBRIDGE
VPMOVZXDQ	xmmreg,xmmrm64	AVX,SANDYBRIDGE
VPMOVZXDQ	Xmmreg，xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMULHUW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMULHUW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMULHRSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMULHRSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMULHW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMULHW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMULLW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
Vmpmullw	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPMULLD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPMULLD	xmmrm128	AVX，SANDYBRIDGE
VPMULUDQ	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE

VPMULUDQ	Xmmreg，xmmreg * ，	AVX，SANDYBRIDGE
	xmmrm128	
	xmmreg,xmmreg*,xmmrm128	
VPMULDQ	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPMULDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPOR	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPOR	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSADBW	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSADBW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSHUFB	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSHUFB	xmmrm128	AVX，SANDYBRIDGE
VPSHUFD	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
VPSHUFD	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPSHUFHW	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
这是什么意思	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
VPSHUFLW	xmmreg,xmmrm128,imm8	AVX,SANDYBRIDGE
(翻译)	Xmmreg，xmmmrm128，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSIGNB	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSIGNB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSIGNW	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSIGNW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSIGND	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSIGND	xmmrm128	AVX，SANDYBRIDGE
VPSLLDQ	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSLLDQ	Xmmreg，xmmreg * ，imm8	AVX，SANDYBRIDGE
VPSRLDQ	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRLDQ	Xmmreg，xmmreg * ，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSLLW	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSLLW	xmmrm128	AVX，SANDYBRIDGE
VPSLLW	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSLLW	Xmmreg，xmmreg * ，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSLLD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VPSLLD	xmmrm128	AVX，SANDYBRIDGE

185

VPSLLD	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSLLD	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSLLQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSLLQ	xmmrm128	AVX，SANDYBRIDGE
VPSLLQ	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSLLQ	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSRAW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSRAW	xmmrm128	AVX，SANDYBRIDGE
VPSRAW	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRAW	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSRAD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSRAD	xmmrm128	AVX，SANDYBRIDGE
VPSRAD	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRAD	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSRLW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSRLW	xmmrm128	AVX，SANDYBRIDGE
VPSRLW	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRLW	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSRLD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSRLD	xmmrm128	AVX，SANDYBRIDGE
VPSRLD	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRLD	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSRLQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSRLQ	xmmrm128	AVX，SANDYBRIDGE
VPSRLQ	xmmreg,xmmreg*,imm8	AVX,SANDYBRIDGE
VPSRLQ	Xmmreg，xmmreg *， imm8	AVX，SANDYBRIDGE
VPTEST	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VPTEST	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VPTEST	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VPTEST	Ymmreg，ymm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBQ	xmmrm128	AVX，SANDYBRIDGE
VPSUBSB	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VPSUBSB	Xmmreg，xmmreg *，	AVX，SANDYBRIDGE

	xmmrm128	
	xmmreg,xmmreg*,xmmrm128	
VPSUBSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBUSB	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBUSB	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPSUBUSW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPSUBUSW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKHBW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKHBW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKHWD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKHWD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKHDQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKHDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKHQDQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKHQDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKLBW	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKLBW	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKLWD	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKLWD	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKLDQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKLDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPUNPCKLQDQ	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPUNPCKLQDQ	xmmrm128	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	
VPXOR	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VPXOR	xmmrm128	AVX，SANDYBRIDGE
VRCPPS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VRCPPS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VRCPPS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VRCPPS	Ymmreg，ymm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VRCPSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VRCPSS	xmmrm32	AVX，SANDYBRIDGE
VRSQRTPS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VRSQRTPS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VRSQRTPS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VRSQRTPS	Ymmreg，ymm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VRSQRTSS	Xmmreg，xmmreg *，	AVX,SANDYBRIDGE
VRSQRTSS	xmmrm32	AVX，SANDYBRIDGE

	xmmreg,xmmrm128,imm8	
VROUNDPD	Xmmreg，xmmmrm128，	AVX,SANDYBRIDGE
VROUNDPD	imm8	AVX，SANDYBRIDGE
VROUNDPD	ymmreg,ymmrm256,imm8	AVX,SANDYBRIDGE
VROUNDPD	Ymmreg，ymmrm256，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmrm128,imm8	
VROUNDPS	Xmmreg，xmmmrm128，	AVX,SANDYBRIDGE
VROUNDPS	imm8	AVX，SANDYBRIDGE
VROUNDPS	ymmreg,ymmrm256,imm8	AVX,SANDYBRIDGE
VROUNDPS	Ymmreg，ymmrm256，imm8	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64,imm8 AVX,SANDYBRIDGE	
VROUNDSD	Xmmreg，xmmreg * ，xmmrm64，imm8 AVX，	
VROUNDSD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm32,imm8 AVX,SANDYBRIDGE	
VROUNDSS	Xmmreg，xmmreg * ，xmmrm32，imm8 AVX，	
VROUNDSS	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VSHUFPD	Xmmreg，xmmreg * ，xmmrm128，imm8 AVX，	
VSHUFPD	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VSHUFPD	Ymmreg，ymmreg * ，ymmrm256，imm8 AVX，	
VSHUFPD	SANDYBRIDGE	
	xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE	
VSHUFPS	Xmmreg，xmmreg * ，xmmrm128，imm8 AVX，	
VSHUFPS	SANDYBRIDGE	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX,SANDYBRIDGE	
VSHUFPS	Ymmreg，ymmreg * ，ymmrm256，imm8 AVX，	
VSHUFPS	SANDYBRIDGE	
VSQRTPD	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VSQRTPD	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VSQRTPD	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VSQRTPD	Ymmreg，ymm256	AVX，SANDYBRIDGE
VSQRTPS	xmmreg,xmmrm128	AVX,SANDYBRIDGE
VSQRTPS	Xmmreg，xmmrm128	AVX，SANDYBRIDGE
VSQRTPS	ymmreg,ymmrm256	AVX,SANDYBRIDGE
VSQRTPS	Ymmreg，ymm256	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	
VSQRTSD	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VSQRTSD	xmmrm64	AVX，SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	
VSQRTSS	Xmmreg，xmmreg * ，	AVX,SANDYBRIDGE
VSQRTSS	xmmrm32	AVX，SANDYBRIDGE

186

		AVX,SANDYBRIDGE
VSTMXCSR	mem32	AVX，
VSTMXCSR	Mem32	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VSUBPD	Xmmreg，xmmreg＊，	AVX，
VSUBPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VSUBPD	Ymmreg，ymmreg＊，	AVX，
VSUBPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VSUBPS	Xmmreg，xmmreg＊，	AVX，
VSUBPS	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VSUBPS	Ymmreg，ymmreg＊，	AVX，
VSUBPS	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm64	AVX,SANDYBRIDGE
VSUBSD	Xmmreg，xmmreg＊，	AVX，
VSUBSD	xmmrm64	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm32	AVX,SANDYBRIDGE
VSUBSS	Xmmreg，xmmreg＊，	AVX，
VSUBSS	xmmrm32	SANDYBRIDGE
		AVX,SANDYBRIDGE
VTESTPS	xmmreg,xmmrm128	AVX，
VTESTPS	Xmmreg，xmmrm128	SANDYBRIDGE
		AVX,SANDYBRIDGE
VTESTPS	ymmreg,ymmrm256	AVX，
VTESTPS	Ymmreg，ymm256	SANDYBRIDGE
		AVX,SANDYBRIDGE
VTESTPD	xmmreg,xmmrm128	AVX，
VTESTPD	Xmmreg，xmmrm128	SANDYBRIDGE
		AVX,SANDYBRIDGE
VTESTPD	ymmreg,ymmrm256	AVX，
VTESTPD	Ymmreg，ymm256	SANDYBRIDGE
		AVX,SANDYBRIDGE
VUCOMISD	xmmreg,xmmrm64	AVX，
VUCOMISD	Xmmreg，xmmrm64	SANDYBRIDGE
		AVX,SANDYBRIDGE
VUCOMISS	xmmreg,xmmrm32	AVX，
VUCOMISS	Xmmrm32	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VUNPCKHPD	Xmmreg，xmmreg＊，	AVX，
VUNPCKHPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VUNPCKHPD	Ymmreg，ymmreg＊，	AVX，
VUNPCKHPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VUNPCKHPS	Xmmreg，xmmreg＊，	AVX，
VUNPCKHPS	xmmrm128	SANDYBRIDGE
VUNPCKHPS	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VUNPCKHPS	Ymmreg，ymmreg＊，	AVX，

	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VUNPCKLPD	Xmmreg，xmmreg＊，	AVX，
VUNPCKLPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VUNPCKLPD	Ymmreg，ymmreg＊，	AVX，
VUNPCKLPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VUNPCKLPS	Xmmreg，xmmreg＊，	AVX，
VUNPCKLPS	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VUNPCKLPS	Ymmreg，ymmreg＊，	AVX，
VUNPCKLPS	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VXORPD	Xmmreg，xmmreg＊，	AVX，
VXORPD	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VXORPD	Ymmreg，ymmreg＊，	AVX，
VXORPD	ymmrm256	SANDYBRIDGE
	xmmreg,xmmreg*,xmmrm128	AVX,SANDYBRIDGE
VXORPS	Xmmreg，xmmreg＊，	AVX，
VXORPS	xmmrm128	SANDYBRIDGE
	ymmreg,ymmreg*,ymmrm256	AVX,SANDYBRIDGE
VXORPS	Ymmreg，ymmreg＊，	AVX，
VXORPS	ymmrm256	SANDYBRIDGE
		AVX,SANDYBRIDGE
VZEROALL		AVX，
VZEROALL		SANDYBRIDGE
		AVX,SANDYBRIDGE
VZEROUPPER		AVX，
VZEROUPPER		SANDYBRIDGE

B.1.27 Intel Carry−Less Multiplication instructions (CLMUL)
英特尔 1.27 无进位乘法指令(CLMUL)

		SSE,WESTMERE
		SSE，
PCLMULLQLQDQ	xmmreg,xmmrm128	WESTMERE 西
PCLMULLQLQDQ	Xmmreg，xmmrm128	米尔
		SSE,WESTMERE
		SSE，
PCLMULHQLQDQ	xmmreg,xmmrm128	WESTMERE 西
PCLMULHQLQDQ	Xmmreg，xmmrm128	米尔
		SSE,WESTMERE
		SSE，
PCLMULLQHQDQ	xmmreg,xmmrm128	WESTMERE 西
PCLMULLQHQDQ	Xmmreg，xmmrm128	米尔
		SSE,WESTMERE
		SSE，
PCLMULHQHQDQ	xmmreg,xmmrm128	WESTMERE 西
Pclmulhqqdq	Xmmreg，xmmrm128	米尔

SSE,WESTMERE

xmmreg,xmmrm128,imm8 SSE，

PCLMULQDQ Xmmreg，xmmmrm128， WESTMERE 西

PCLMULQDQ imm8 米尔

B.1.28 Intel AVX Carry−Less Multiplication instructions (CLMUL)

B. 1.28 英特尔 AVX 无进位乘法指令(CLMUL)

xmmreg,xmmreg*,xmmrm128

VPCLMULLQLQDQ Xmmreg，xmmreg *， AVX,SANDYBRIDGE

VPCLMULLQLQDQ xmmrm128 AVX，SANDYBRIDGE

xmmreg,xmmreg*,xmmrm128

VPCLMULHQLQDQ Xmmreg，xmmreg *， AVX,SANDYBRIDGE

VPCLMULHQLQDQ xmmrm128 AVX，SANDYBRIDGE

xmmreg,xmmreg*,xmmrm128

VPCLMULLQHQDQ Xmmreg，xmmreg *， AVX,SANDYBRIDGE

Qhqdq xmmrm128 AVX，SANDYBRIDGE

xmmreg,xmmreg*,xmmrm128

VPCLMULHQHQDQ Xmmreg，xmmreg *， AVX,SANDYBRIDGE

VPCLMULHQHQDQ xmmrm128 AVX，SANDYBRIDGE

xmmreg,xmmreg*,xmmrm128,imm8 AVX,SANDYBRIDGE

VPCLMULQDQ Xmmreg，xmmreg *，xmmrm128，imm8 AVX，

VPCLMULQDQ SANDYBRIDGE

ymmreg,ymmreg*,ymmrm256

VPCLMULLQLQDQ Ymmreg，ymmreg *， VPCLMULQDQ

VPCLMULLQLQDQ ymmrm256 VPCLMULQDQ

ymmreg,ymmreg*,ymmrm256

VPCLMULHQLQDQ Ymmreg，ymmreg *， VPCLMULQDQ

VPCLMULHQLQDQ ymmrm256 VPCLMULQDQ

ymmreg,ymmreg*,ymmrm256

VPCLMULLQHQDQ Ymmreg，ymmreg *， VPCLMULQDQ

Qhqdq ymmrm256 VPCLMULQDQ

ymmreg,ymmreg*,ymmrm256

VPCLMULHQHQDQ Ymmreg，ymmreg *， VPCLMULQDQ

VPCLMULHQHQDQ ymmrm256 VPCLMULQDQ

VPCLMULQDQ ymmreg,ymmreg*,ymmrm256,imm8 VPCLMULQDQ

VPCLMULQDQ Ymmreg，ymmreg *，ymmrm256，imm8 VPCLMULQDQ

xmmreg,xmmreg*,xmmrm128

VPCLMULLQLQDQ Xmmreg，xmmreg *， AVX512VL,VPCLMULQDQ

VPCLMULLQLQDQ xmmrm128 AVX512VL，VPCLMULQDQ

xmmreg,xmmreg*,xmmrm128

VPCLMULHQLQDQ Xmmreg，xmmreg *， AVX512VL,VPCLMULQDQ

VPCLMULHQLQDQ xmmrm128 AVX512VL，VPCLMULQDQ

xmmreg,xmmreg*,xmmrm128

VPCLMULLQHQDQ Xmmreg，xmmreg *， AVX512VL,VPCLMULQDQ

Qhqdq xmmrm128 AVX512VL，VPCLMULQDQ

xmmreg,xmmreg*,xmmrm128

VPCLMULHQHQDQ Xmmreg，xmmreg *， AVX512VL,VPCLMULQDQ

VPCLMULHQHQDQ xmmrm128 AVX512VL，VPCLMULQDQ

VPCLMULQDQ xmmreg,xmmreg*,xmmrm128,imm8 AVX512VL,VPCLMULQDQ

VPCLMULQDQ Xmmreg，xmmreg *，xmmrm128，imm8 AVX512VL，

	VPCLMULQDQ	
	ymmreg,ymmreg*,ymmrm256	
VPCLMULLQLQDQ	Ymmreg， ymmreg *，	AVX512VL,VPCLMULQDQ
VPCLMULLQLQDQ	ymmrm256	AVX512VL， VPCLMULQDQ
	ymmreg,ymmreg*,ymmrm256	
VPCLMULHQLQDQ	Ymmreg， ymmreg *，	AVX512VL,VPCLMULQDQ
VPCLMULHQLQDQ	ymmrm256	AVX512VL， VPCLMULQDQ

187

	ymmreg,ymmreg*,ymmrm256	
VPCLMULLQHQDQ	Ymmreg，ymmreg *，	AVX512VL,VPCLMULQDQ
Qhqdq	ymmrm256	AVX512VL，VPCLMULQDQ
	ymmreg,ymmreg*,ymmrm256	
VPCLMULHQHQDQ	Ymmreg，ymmreg *，	AVX512VL,VPCLMULQDQ
VPCLMULHQHQDQ	ymmrm256	AVX512VL，VPCLMULQDQ
	ymmreg,ymmreg*,ymmrm256,imm8 AVX512VL,VPCLMULQDQ	
VPCLMULQDQ	Ymmreg，ymmreg *， ymmrm256，imm8 AVX512VL，	
VPCLMULQDQ	VPCLMULQDQ	
	zmmreg,zmmreg*,zmmrm512	
VPCLMULLQLQDQ	Zmmreg，zmmreg *，	AVX512,VPCLMULQDQ
VPCLMULLQLQDQ	zmmrm512	AVX512，VPCLMULQDQ
	zmmreg,zmmreg*,zmmrm512	
VPCLMULHQLQDQ	Zmmreg，zmmreg *，	AVX512,VPCLMULQDQ
VPCLMULHQLQDQ	zmmrm512	AVX512，VPCLMULQDQ
	zmmreg,zmmreg*,zmmrm512	
VPCLMULLQHQDQ	Zmmreg，zmmreg *，	AVX512,VPCLMULQDQ
Qhqdq	zmmrm512	AVX512，VPCLMULQDQ
	zmmreg,zmmreg*,zmmrm512	
VPCLMULHQHQDQ	Zmmreg，zmmreg *，	AVX512,VPCLMULQDQ
VPCLMULHQHQDQ	zmmrm512	AVX512，VPCLMULQDQ
	zmmreg,zmmreg*,zmmrm512,imm8 AVX512,VPCLMULQDQ	
VPCLMULQDQ	Zmmreg，zmmreg *， zmmrm512，imm8 AVX512，	
VPCLMULQDQ	VPCLMULQDQ	

B.1.29 Intel Fused Multiply−Add instructions (FMA)
英特尔融合乘加指令(FMA)

		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADD132PS	Xmmreg，xmmreg，	FM
VFMADD132PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD132PS	Ymmreg，ymmreg，	FM
VFMADD132PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADD132PD	Xmmreg，xmmreg，	FM
VFMADD132PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD132PD	Ymmreg，ymmreg，	FM
VFMADD132PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADD312PS	Xmmreg，xmmreg，	FM
VFMADD312PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD312PS	Ymmreg，ymmreg，	FM
VFMADD312PS	ymmrm256	A

	xmmreg,xmmreg,xmmrm128	FM A
VFMADD312PD VFMADD312PD	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD312PD VFMADD312PD	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD213PS VFMADD213PS	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD213PS VFMADD213PS	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD213PD VFMADD213PD	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD213PD VFMADD213PD	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD123PS VFMADD123PS	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD123PS VFMADD123PS	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD123PD VFMADD123PD	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD123PD VFMADD123PD	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD231PS VFMADD231PS	Xmmreg，xmmreg，xmmrm128	FM A
	ymmreg,ymmreg,ymmrm256	FM A
VFMADD231PS VFMADD231PS	Ymmreg，ymmreg，ymmrm256	FM A
	xmmreg,xmmreg,xmmrm128	FM A
VFMADD231PD VFMADD231PD	Xmmreg，xmmreg，xmmrm128	FM A

		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD231PD	Ymmreg，ymmreg，	FM
VFMADD231PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADD321PS	Xmmreg，xmmreg，	FM
VFMADD321PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD321PS	Ymmreg，ymmreg，	FM
VFMADD321PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADD321PD	Xmmreg，xmmreg，	FM
VFMADD321PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADD321PD	Ymmreg，ymmreg，	FM
VFMADD321PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB132PS	Xmmreg，xmmreg，	FM
VFMADDSUB132PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB132PS	Ymmreg，ymmreg，	FM
VFMADDSUB132PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB132PD	Xmmreg，xmmreg，	FM
VFMADDSUB132PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB132PD	Ymmreg，ymmreg，	FM
VFMADDSUB132PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB312PS	Xmmreg，xmmreg，	FM
VFMADDSUB312PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB312PS	Ymmreg，ymmreg，	FM
VFMADDSUB312PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB312PD	Xmmreg，xmmreg，	FM
VFMADDSUB312PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB312PD	Ymmreg，ymmreg，	FM
VFMADDSUB312PD	ymmrm256	A

		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB213PS	Xmmreg，xmmreg，	FM
VFMADDSUB213PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB213PS	Ymmreg，ymmreg，	FM
VFMADDSUB213PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB213PD	Xmmreg，xmmreg，	FM
VFMADDSUB213PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB213PD	Ymmreg，ymmreg，	FM
VFMADDSUB213PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB123PS	Xmmreg，xmmreg，	FM
VFMADDSUB123PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB123PS	Ymmreg，ymmreg，	FM
VFMADDSUB123PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB123PD	Xmmreg，xmmreg，	FM
VFMADDSUB123PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB123PD	Ymmreg，ymmreg，	FM
VFMADDSUB123PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB231PS	Xmmreg，xmmreg，	FM
VFMADDSUB231PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMADDSUB231PS	Ymmreg，ymmreg，	FM
VFMADDSUB231PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMADDSUB231PD	Xmmreg，xmmreg，	FM
VFMADDSUB231PD	xmmrm128	A

188

	ymmreg,ymmreg,ymmrm256	FMA
VFMADDSUB231PD VFMADDSUB231PD	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMADDSUB321PS VFMADDSUB321PS	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMADDSUB321PS VFMADDSUB321PS	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMADDSUB321PD VFMADDSUB321PD	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMADDSUB321PD VFMADDSUB321PD	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMSUB132PS VFMSUB132PS	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB132PS VFMSUB132PS	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMSUB132PD VFMSUB132PD	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB132PD VFMSUB132PD	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMSUB312PS VFMSUB312PS	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB312PS VFMSUB312PS	Ymmreg，ymmreg， ymmrm256	FMA
	xmmreg,xmmreg,xmmrm128	FMA
VFMSUB312PD VFMSUB312PD	Xmmreg，xmmreg， xmmrm128	FMA
	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB312PD VFMSUB312PD	Ymmreg，ymmreg， ymmrm256	FMA

		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB213PS	Xmmreg，xmmreg，	FM
VFMSUB213PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB213PS	Ymmreg，ymmreg，	FM
VFMSUB213PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB213PD	Xmmreg，xmmreg，	FM
VFMSUB213PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB213PD	Ymmreg，ymmreg，	FM
VFMSUB213PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB123PS	Xmmreg，xmmreg，	FM
VFMSUB123PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB123PS	Ymmreg，ymmreg，	FM
VFMSUB123PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB123PD	Xmmreg，xmmreg，	FM
VFMSUB123PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB123PD	Ymmreg，ymmreg，	FM
VFMSUB123PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB231PS	Xmmreg，xmmreg，	FM
VFMSUB231PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB231PS	Ymmreg，ymmreg，	FM
VFMSUB231PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB231PD	Xmmreg，xmmreg，	FM
VFMSUB231PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFMSUB231PD	Ymmreg，ymmreg，	FM
VFMSUB231PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFMSUB321PS	Xmmreg，xmmreg，	FM
VFMSUB321PS	xmmrm128	A

VFMSUB321PS	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB321PS	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUB321PD	xmmreg,xmmreg,xmmrm128	FMA
VFMSUB321PD	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUB321PD	ymmreg,ymmreg,ymmrm256	FMA
VFMSUB321PD	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUBADD132PS	xmmreg,xmmreg,xmmrm128	FMA
VFMSUBADD132PS	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUBADD132PS	ymmreg,ymmreg,ymmrm256	FMA
VFMSUBADD132PS	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUBADD132PD	xmmreg,xmmreg,xmmrm128	FMA
VFMSUBADD132PD	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUBADD132PD	ymmreg,ymmreg,ymmrm256	FMA
VFMSUBADD132PD	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUBADD312PS	xmmreg,xmmreg,xmmrm128	FMA
VFMSUBADD312PS	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUBADD312PS	ymmreg,ymmreg,ymmrm256	FMA
VFMSUBADD312PS	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUBADD312PD	xmmreg,xmmreg,xmmrm128	FMA
VFMSUBADD312PD	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUBADD312PD	ymmreg,ymmreg,ymmrm256	FMA
VFMSUBADD312PD	Ymmreg，ymmreg，ymmrm256	FMA
VFMSUBADD213PS	xmmreg,xmmreg,xmmrm128	FMA
VFMSUBADD213PS	Xmmreg，xmmreg，xmmrm128	FMA
VFMSUBADD213PS	ymmreg,ymmreg,ymmrm256	FMA
VFMSUBADD213PS	Ymmreg，ymmreg，ymmrm256	FMA

VFMSUBADD213PD VFMSUBADD213PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD213PD VFMSUBADD213PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD123PS VFMSUBADD123PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD123PS VFMSUBADD123PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD123PD VFMSUBADD123PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD123PD VFMSUBADD123PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD231PS VFMSUBADD231PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD231PS VFMSUBADD231PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD231PD VFMSUBADD231PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD231PD VFMSUBADD231PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD321PS VFMSUBADD321PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA
VFMSUBADD321PS VFMSUBADD321PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg，ymmrm256	FMA FMA
VFMSUBADD321PD VFMSUBADD321PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg，xmmrm128	FMA FMA

	ymmreg,ymmreg,ymmrm256	FM A
VFMSUBADD321PD	Ymmreg，ymmreg，	FM
VFMSUBADD321PD	ymmrm256	A
	xmmreg,xmmreg,xmmrm128	FM A
VFNMADD132PS	Xmmreg，xmmreg，	FM
VFNMADD132PS	xmmrm128	A

189

VFNMADD132PS VFNMADD132PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD132PD VFNMADD132PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD132PD VFNMADD132PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD312PS VFNMADD312PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD312PS VFNMADD312PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD312PD VFNMADD312PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD312PD VFNMADD312PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD213PS VFNMADD213PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD213PS VFNMADD213PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD213PD VFNMADD213PD	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD213PD VFNMADD213PD	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A
VFNMADD123PS VFNMADD123PS	xmmreg,xmmreg,xmmrm128 Xmmreg，xmmreg， xmmrm128	FM A FM A
VFNMADD123PS VFNMADD123PS	ymmreg,ymmreg,ymmrm256 Ymmreg，ymmreg， ymmrm256	FM A FM A

		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMADD123PD	Xmmreg， xmmreg，	FM
VFNMADD123PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMADD123PD	Ymmreg， ymmreg，	FM
VFNMADD123PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMADD231PS	Xmmreg， xmmreg，	FM
VFNMADD231PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMADD231PS	Ymmreg， ymmreg，	FM
VFNMADD231PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMADD231PD	Xmmreg， xmmreg，	FM
VFNMADD231PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMADD231PD	Ymmreg， ymmreg，	FM
VFNMADD231PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMADD321PS	Xmmreg， xmmreg，	FM
VFNMADD321PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMADD321PS	Ymmreg， ymmreg，	FM
VFNMADD321PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMADD321PD	Xmmreg， xmmreg，	FM
VFNMADD321PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMADD321PD	Ymmreg， ymmreg，	FM
VFNMADD321PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB132PS	Xmmreg， xmmreg，	FM
VFNMSUB132PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB132PS	Ymmreg， ymmreg，	FM
VFNMSUB132PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB132PD	Xmmreg， xmmreg，	FM
VFNMSUB132PD	xmmrm128	A

		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB132PD	Ymmreg，ymmreg，	FM
VFNMSUB132PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB312PS	Xmmreg，xmmreg，	FM
VFNMSUB312PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB312PS	Ymmreg，ymmreg，	FM
VFNMSUB312PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB312PD	Xmmreg，xmmreg，	FM
VFNMSUB312PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB312PD	Ymmreg，ymmreg，	FM
VFNMSUB312PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB213PS	Xmmreg，xmmreg，	FM
VFNMSUB213PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB213PS	Ymmreg，ymmreg，	FM
VFNMSUB213PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB213PD	Xmmreg，xmmreg，	FM
VFNMSUB213PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB213PD	Ymmreg，ymmreg，	FM
VFNMSUB213PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB123PS	Xmmreg，xmmreg，	FM
VFNMSUB123PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB123PS	Ymmreg，ymmreg，	FM
VFNMSUB123PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB123PD	Xmmreg，xmmreg，	FM
VFNMSUB123PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB123PD	Ymmreg，ymmreg，	FM
VFNMSUB123PD	ymmrm256	A

		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB231PS	Xmmreg，xmmreg，	FM
VFNMSUB231PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB231PS	Ymmreg，ymmreg，	FM
VFNMSUB231PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB231PD	Xmmreg，xmmreg，	FM
VFNMSUB231PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB231PD	Ymmreg，ymmreg，	FM
VFNMSUB231PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB321PS	Xmmreg，xmmreg，	FM
VFNMSUB321PS	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB321PS	Ymmreg，ymmreg，	FM
VFNMSUB321PS	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm128	A
VFNMSUB321PD	Xmmreg，xmmreg，	FM
VFNMSUB321PD	xmmrm128	A
		FM
	ymmreg,ymmreg,ymmrm256	A
VFNMSUB321PD	Ymmreg，ymmreg，	FM
VFNMSUB321PD	ymmrm256	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFMADD132SS	Xmmreg，xmmreg，	FM
VFMADD132SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFMADD132SD	Xmmreg，xmmreg，	FM
VFMADD132SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFMADD312SS	Xmmreg，xmmreg，	FM
VFMADD312SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFMADD312SD	Xmmreg，xmmreg，	FM
VFMADD312SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFMADD213SS	Xmmreg，xmmreg，	FM
VFMADD213SS	xmmrm32	A

		FM
	xmmreg,xmmreg,xmmrm64	A
VFMADD213SD	Xmmreg，xmmreg，	FM
VFMADD213SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFMADD123SS	Xmmreg，xmmreg，	FM
VFMADD123SS	xmmrm32	A

190

	xmmreg,xmmreg,xmmrm64	FM A
VFMADD123SD VFMADD123SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMADD231SS VFMADD231SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMADD231SD VFMADD231SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMADD321SS VFMADD321SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMADD321SD VFMADD321SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMSUB132SS VFMSUB132SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMSUB132SD VFMSUB132SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMSUB312SS VFMSUB312SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMSUB312SD VFMSUB312SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMSUB213SS VFMSUB213SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMSUB213SD VFMSUB213SD	Xmmreg，xmmreg， xmmrm64	FM A
	xmmreg,xmmreg,xmmrm32	FM A
VFMSUB123SS VFMSUB123SS	Xmmreg，xmmreg， xmmrm32	FM A
	xmmreg,xmmreg,xmmrm64	FM A
VFMSUB123SD VFMSUB123SD	Xmmreg，xmmreg， xmmrm64	FM A

		FM
	xmmreg,xmmreg,xmmrm32	A
VFMSUB231SS	Xmmreg，xmmreg，	FM
VFMSUB231SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFMSUB231SD	Xmmreg，xmmreg，	FM
VFMSUB231SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFMSUB321SS	Xmmreg，xmmreg，	FM
VFMSUB321SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFMSUB321SD	Xmmreg，xmmreg，	FM
VFMSUB321SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD132SS	Xmmreg，xmmreg，	FM
VFNMADD132SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD132SD	Xmmreg，xmmreg，	FM
VFNMADD132SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD312SS	Xmmreg，xmmreg，	FM
VFNMADD312SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD312SD	Xmmreg，xmmreg，	FM
VFNMADD312SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD213SS	Xmmreg，xmmreg，	FM
VFNMADD213SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD213SD	Xmmreg，xmmreg，	FM
VFNMADD213SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD123SS	Xmmreg，xmmreg，	FM
VFNMADD123SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD123SD	Xmmreg，xmmreg，	FM
VFNMADD123SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD231SS	Xmmreg，xmmreg，	FM
VFNMADD231SS	xmmrm32	A

		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD231SD	Xmmreg，xmmreg，	FM
VFNMADD231SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMADD321SS	Xmmreg，xmmreg，	FM
VFNMADD321SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMADD321SD	Xmmreg，xmmreg，	FM
VFNMADD321SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB132SS	Xmmreg，xmmreg，	FM
VFNMSUB132SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB132SD	Xmmreg，xmmreg，	FM
VFNMSUB132SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB312SS	Xmmreg，xmmreg，	FM
VFNMSUB312SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB312SD	Xmmreg，xmmreg，	FM
VFNMSUB312SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB213SS	Xmmreg，xmmreg，	FM
VFNMSUB213SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB213SD	Xmmreg，xmmreg，	FM
VFNMSUB213SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB123SS	Xmmreg，xmmreg，	FM
VFNMSUB123SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB123SD	Xmmreg，xmmreg，	FM
VFNMSUB123SD	xmmrm64	A
		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB231SS	Xmmreg，xmmreg，	FM
VFNMSUB231SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB231SD	Xmmreg，xmmreg，	FM
VFNMSUB231SD	xmmrm64	A

		FM
	xmmreg,xmmreg,xmmrm32	A
VFNMSUB321SS	Xmmreg，xmmreg，	FM
VFNMSUB321SS	xmmrm32	A
		FM
	xmmreg,xmmreg,xmmrm64	A
VFNMSUB321SD	Xmmreg，xmmreg，	FM
VFNMSUB321SD	xmmrm64	A

B.1.30 Intel post−32 nm processor instructions
B. 1.30 英特尔后 32 纳米处理器指令

		LON
RDFSBASE	reg32	G
RDFSBASE	Reg32	龙
		LON
RDFSBASE	reg64	G
RDFSBASE	Reg64	龙
		LON
RDGSBASE	reg32	G
RDGSBASE	Reg32	龙
		LON
RDGSBASE	reg64	G
RDGSBASE	Reg64	龙
RDRAND	reg16	
RDRAND	规例 16	
RDRAND	reg32	
RDRAND	Reg32	
		LON
RDRAND	reg64	G
RDRAND	Reg64	龙
		LON
WRFSBASE	reg32	G
WRFSBASE	Reg32	龙
		LON
WRFSBASE	reg64	G
WRFSBASE	Reg64	龙
		LON
WRGSBASE	reg32	G
WRGSBASE	Reg32	龙

191

		LON
WRGSBASE	reg64	G
WRGSBASE	Reg64	龙
VCVTPH2PS		
VCVTPH2PS	ymmreg,xmmrm128	AVX
VCVTPH2PS	Ymmreg，xmmrm128	AVX
VCVTPH2PS		
VCVTPH2PS	xmmreg,xmmrm64	AVX
VCVTPH2PS	Xmmreg，xmmrm64	AVX
	xmmrm128,ymmreg,imm8	
VCVTPS2PH	Xmmrm128，ymmreg，	AVX
VCVTPS2PH	imm8	AVX
VCVTPS2PH	xmmrm64,xmmreg,imm8	AVX
VCVTPS2PH	Xmmrm64，xmreg，imm8	AVX
ADCX	reg32,rm32	
ADCX	Reg32，rm32	
		LON
ADCX	reg64,rm64	G
ADCX	Reg64，rm64	龙
ADOX	reg32,rm32	
ADOX	Reg32，rm32	
		LON
ADOX	reg64,rm64	G
ADOX	Reg64，rm64	龙
RDSEED	reg16	
RDSEED	规例 16	
RDSEED	reg32	
RDSEED	Reg32	
		LON
RDSEED	reg64	G
RDSEED	Reg64	龙
CLAC		PRIV
CLAC		PRIV
STAC		
战术空中支援中		PRIV
心		PRIV

B.1.31 VIA (Centaur) security instructions
B. 1.31 VIA (Centaur)安全指示

XSTORE	PENT,CYRIX
XSTORE PENT，CYRIX	
XCRYPTECB	PENT,CYRIX
XCRYPTECB pented，CYRIX	
XCRYPTCBC	PENT,CYRIX
XCRYPTCBC pented，CYRIX	
XCRYPTCTR	PENT,CYRIX
XCRYPTCTR pented，CYRIX	
XCRYPTCFB	PENT,CYRIX
XCRYPTCFB pented，CYRIX	
XCRYPTOFB	PENT,CYRIX
XCRYPTOFB 暂停，CYRIX	
MONTMUL	PENT,CYRIX

```
MONTMUL pented, CYRIX
XSHA1                                             PENT,CYRIX
Xsha1 被压抑的 CYRIX
XSHA256                                           PENT,CYRIX
XSHA256 pented, CYRIX
```

B.1.32 AMD Lightweight Profiling (LWP) instructions
1.32 AMD 轻量级分析(LWP)说明书

LLWPCB	reg32	AMD,386
LLWPCB	Reg32	AMD，386
LLWPCB	reg64	AMD,X64
LLWPCB	Reg64	AMD X64
SLWPCB	reg32	AMD,386
SLWPCB	Reg32	AMD，386
SLWPCB	reg64	AMD,X64
SLWPCB	Reg64	AMD X64
LWPVAL	reg32,rm32,imm32	AMD,386
LWPVAL	Reg32，rm32，imm32	AMD，386
LWPVAL	reg64,rm32,imm32	AMD,X64
LWPVAL	Reg64，rm32，imm32	AMD X64
LWPINS	reg32,rm32,imm32	AMD,386
LWPINS	Reg32，rm32，imm32	AMD，386
LWPINS	reg64,rm32,imm32	AMD,X64
LWPINS	Reg64，rm32，imm32	AMD X64

B.1.33 AMD XOP and FMA4 instructions (SSE5)
B. 1.33 AMD XOP 和 fma4 指令(SSE5)

```
VFMADDPD          xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMADDPD xmmreg, xmmreg * , xmmrm128, xmmreg AMD, SSE5
VFMADDPD          ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMADDPD ymmreg, ymmreg * , ymmrm256, ymmreg AMD, SSE5
VFMADDPD          xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMADDPD xmmreg, xmmreg * , xmmreg, xmmrm128 AMD, SSE5
VFMADDPD          ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMADDPD ymmreg, ymmreg * , ymmreg, ymmrm256 AMD, SSE5
VFMADDPS          xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMADDPS xmmreg, xmmreg * , xmmrm128, xmmreg AMD, SSE5
VFMADDPS          ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMADDPS ymmreg, ymmreg * , ymmrm256, ymmreg AMD, SSE5
VFMADDPS          xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMADDPS xmmreg, xmmreg * , xmmreg, xmmrm128 AMD, SSE5
VFMADDPS          ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMADDPS ymmreg, ymmreg * , ymmreg, ymmrm256 AMD, SSE5
VFMADDSD          xmmreg,xmmreg*,xmmrm64,xmmreg AMD,SSE5
VFMADDSD xmmreg, xmmreg * , xmmrm64, xmmreg AMD, SSE5
VFMADDSD          xmmreg,xmmreg*,xmmreg,xmmrm64 AMD,SSE5
VFMADDSD xmmreg, xmmreg * , xmmreg, xmmrm64 AMD, SSE5
VFMADDSS          xmmreg,xmmreg*,xmmrm32,xmmreg AMD,SSE5
VFMADDSS xmmreg, xmmreg * , xmmrm32, xmmreg AMD, SSE5
VFMADDSS          xmmreg,xmmreg*,xmmreg,xmmrm32 AMD,SSE5
VFMADDSS xmmreg, xmmreg * , xmmreg, xmmrm32 AMD, SSE5
VFMADDSUBPD       xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMADDSUBPD xmmreg, xmmreg * , xmmrm128, xmmreg AMD, SSE5
VFMADDSUBPD       ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMADDSUBPD ymmreg, ymmreg * , ymmrm256, ymmreg AMD, SSE5
VFMADDSUBPD       xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMADDSUBPD xmmreg, xmmreg * , xmmreg, xmmrm128 AMD, SSE5
VFMADDSUBPD       ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMADDSUBPD ymmreg, ymmreg * , ymmreg, ymmrm256 AMD, SSE5
```

192

VFMADDSUBPS	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMADDSUBPS	Xmmreg，xmmreg *，xmmrm128，xmmreg
VFMADDSUBPS	AMD，SSE5
VFMADDSUBPS	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMADDSUBPS	Ymmreg，ymmreg *，ymmrm256，ymmreg
VFMADDSUBPS	AMD，SSE5
VFMADDSUBPS	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMADDSUBPS	Xmmreg，xmmreg *，xmmreg，xmmrm128
VFMADDSUBPS	AMD，SSE5
VFMADDSUBPS	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMADDSUBPS	Ymmreg，ymmreg *，ymmreg，ymmrm256
VFMADDSUBPS	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMSUBADDPD	Xmmreg，xmmreg *，xmmrm128，xmmreg
VFMSUBADDPD	AMD，SSE5
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMSUBADDPD	Ymmreg，ymmreg *，ymmrm256，ymmreg
VFMSUBADDPD	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMSUBADDPD	Xmmreg，xmmreg *，xmmreg，xmmrm128
VFMSUBADDPD	AMD，SSE5
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMSUBADDPD	Ymmreg，ymmreg *，ymmreg，ymmrm256
VFMSUBADDPD	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMSUBADDPS	Xmmreg，xmmreg *，xmmrm128，xmmreg
VFMSUBADDPS	AMD，SSE5
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMSUBADDPS	Ymmreg，ymmreg *，ymmrm256，ymmreg
VFMSUBADDPS	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMSUBADDPS	Xmmreg，xmmreg *，xmmreg，xmmrm128
VFMSUBADDPS	AMD，SSE5
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMSUBADDPS	Ymmreg，ymmreg *，ymmreg，ymmrm256
VFMSUBADDPS	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFMSUBPD	Xmmreg，xmmreg *，xmmrm128，xmmreg
VFMSUBPD	AMD，SSE5
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMSUBPD	Ymmreg，ymmreg *，ymmrm256，ymmreg
VFMSUBPD	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMSUBPD	Xmmreg，xmmreg *，xmmreg，xmmrm128
VFMSUBPD	AMD，SSE5
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMSUBPD	Ymmreg，ymmreg *，ymmreg，ymmrm256
VFMSUBPD	AMD，SSE5
VFMSUBPS	
VFMSUBPS	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5

VFMSUBPS	Xmmreg，xmmreg＊，xmmrm128，xmmreg
	AMD，SSE5
VFMSUBPS	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFMSUBPS	Ymmreg，ymmreg＊，ymmrm256，ymmreg
VFMSUBPS	AMD，SSE5
VFMSUBPS	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFMSUBPS	Xmmreg，xmmreg＊，xmmreg，xmmrm128
VFMSUBPS	AMD，SSE5
VFMSUBPS	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFMSUBPS	Ymmreg，ymmreg＊，ymmreg，ymmrm256
VFMSUBPS	AMD，SSE5
	xmmreg,xmmreg*,xmmrm64,xmmreg AMD,SSE5
VFMSUBSD	Xmmreg，xmmreg＊，xmmrm64，xmmreg
VFMSUBSD	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm64 AMD,SSE5
VFMSUBSD	Xmmreg，xmmreg＊，xmmreg，xmmrm64
VFMSUBSD	AMD，SSE5
	xmmreg,xmmreg*,xmmrm32,xmmreg AMD,SSE5
VFMSUBSS	Xmmreg，xmmreg＊，xmmrm32，xmmreg
VFMSUBSS	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm32 AMD,SSE5
VFMSUBSS	Xmmreg，xmmreg＊，xmmreg，xmmrm32
VFMSUBSS	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFNMADDPD	Xmmreg，xmmreg＊，xmmrm128，xmmreg
VFNMADDPD	AMD，SSE5
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFNMADDPD	Ymmreg，ymmreg＊，ymmrm256，ymmreg
VFNMADDPD	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFNMADDPD	Xmmreg，xmmreg＊，xmmreg，xmmrm128
VFNMADDPD	AMD，SSE5
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFNMADDPD	Ymmreg，ymmreg＊，ymmreg，ymmrm256
VFNMADDPD	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5
VFNMADDPS	Xmmreg，xmmreg＊，xmmrm128，xmmreg
VFNMADDPS	AMD，SSE5
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5
VFNMADDPS	Ymmreg，ymmreg＊，ymmrm256，ymmreg
VFNMADDPS	AMD，SSE5
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5
VFNMADDPS	Xmmreg，xmmreg＊，xmmreg，xmmrm128
VFNMADDPS	AMD，SSE5
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5
VFNMADDPS	Ymmreg，ymmreg＊，ymmreg，ymmrm256
VFNMADDPS	AMD，SSE5
	xmmreg,xmmreg*,xmmrm64,xmmreg AMD,SSE5
VFNMADDSD	Xmmreg，xmmreg＊，xmmrm64，xmmreg
VFNMADDSD	AMD，SSE5

	xmmreg,xmmreg*,xmmreg,xmmrm64 AMD,SSE5	
VFNMADDSD	Xmmreg，xmmreg *，xmmreg，xmmrm64	
VFNMADDSD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm32,xmmreg AMD,SSE5	
VFNMADDSS	Xmmreg，xmmreg *，xmmrm32，xmmreg	
VFNMADDSS	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm32 AMD,SSE5	
VFNMADDSS	Xmmreg，xmmreg *，xmmreg，xmmrm32	
VFNMADDSS	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VFNMSUBPD	Xmmreg，xmmreg *，xmmrm128，xmmreg	
VFNMSUBPD	AMD，SSE5	
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5	
VFNMSUBPD	Ymmreg，ymmreg *，ymmrm256，ymmreg	
VFNMSUBPD	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5	
VFNMSUBPD	Xmmreg，xmmreg *，xmmreg，xmmrm128	
VFNMSUBPD	AMD，SSE5	
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5	
VFNMSUBPD	Ymmreg，ymmreg *，ymmreg，ymmrm256	
VFNMSUBPD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VFNMSUBPS	Xmmreg，xmmreg *，xmmrm128，xmmreg	
VFNMSUBPS	AMD，SSE5	
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5	
VFNMSUBPS	Ymmreg，ymmreg *，ymmrm256，ymmreg	
VFNMSUBPS	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5	
VFNMSUBPS	Xmmreg，xmmreg *，xmmreg，xmmrm128	
VFNMSUBPS	AMD，SSE5	
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5	
VFNMSUBPS	Ymmreg，ymmreg *，ymmreg，ymmrm256	
VFNMSUBPS	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm64,xmmreg AMD,SSE5	
VFNMSUBSD	Xmmreg，xmmreg *，xmmrm64，xmmreg	
VFNMSUBSD	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm64 AMD,SSE5	
VFNMSUBSD	Xmmreg，xmmreg *，xmmreg，xmmrm64	
VFNMSUBSD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm32,xmmreg AMD,SSE5	
VFNMSUBSS	Xmmreg，xmmreg *，xmmrm32，xmmreg	
VFNMSUBSS	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm32 AMD,SSE5	
VFNMSUBSS	Xmmreg，xmmreg *，xmmreg，xmmrm32	
VFNMSUBSS	AMD，SSE5	
VFRCZPD	xmmreg,xmmrm128*	AMD,SSE5
VFRCZPD	Xmmreg，xmmrm128 *	AMD，SSE5
VFRCZPD	ymmreg,ymmrm256*	AMD,SSE5
VFRCZPD	Ymmrm256 *	AMD，SSE5
VFRCZPS	xmmreg,xmmrm128*	AMD,SSE5

VFRCZPS	Xmmreg，xmmrm128 *	AMD，SSE5
VFRCZPS	ymmreg,ymmrm256*	AMD,SSE5
VFRCZPS	Ymmrm256 *	AMD，SSE5
VFRCZSD	xmmreg,xmmrm64*	AMD,SSE5
VFRCZSD	Xmmrm64 *	AMD，SSE5
VFRCZSS	xmmreg,xmmrm32*	AMD,SSE5
VFRCZSS	Xmmrreg，xmmrm32 *	AMD，SSE5

193

	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPCMOV	Xmmreg，xmmreg *，xmmrm128，xmmreg	
VPCMOV	AMD，SSE5	
	ymmreg,ymmreg*,ymmrm256,ymmreg AMD,SSE5	
VPCMOV	Ymmreg，ymmreg *，ymmrm256，ymmreg	
VPCMOV	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5	
VPCMOV	Xmmreg，xmmreg *，xmmreg，xmmrm128	
VPCMOV	AMD，SSE5	
	ymmreg,ymmreg*,ymmreg,ymmrm256 AMD,SSE5	
VPCMOV	Ymmreg，ymmreg *，ymmreg，ymmrm256	
VPCMOV	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMB	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMB	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMD	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMD	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMQ	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMQ	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMUB	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMUB	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMUD	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMUD	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMUQ	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMUQ	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMUW	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMUW	SSE5	
	xmmreg,xmmreg*,xmmrm128,imm8 AMD,SSE5	
VPCOMW	Xmmreg，xmmreg *，xmmrm128，imm8 AMD，	
VPCOMW	SSE5	
VPHADDBD	xmmreg,xmmrm128*	AMD,SSE5
VPHADDBD	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDBQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDBQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDBW	xmmreg,xmmrm128*	AMD,SSE5
VPHADDBW	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDDQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDDQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDUBD	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUBD	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDUBQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUBQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDUBW	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUBW	Xmmreg，xmmrm128 *	AMD，SSE5

VPHADDUDQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUDQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDUWD	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUWD	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDUWQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDUWQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDWD	xmmreg,xmmrm128*	AMD,SSE5
VPHADDWD	Xmmreg，xmmrm128 *	AMD，SSE5
VPHADDWQ	xmmreg,xmmrm128*	AMD,SSE5
VPHADDWQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHSUBBW	xmmreg,xmmrm128*	AMD,SSE5
VPHSUBBW	Xmmreg，xmmrm128 *	AMD，SSE5
VPHSUBDQ	xmmreg,xmmrm128*	AMD,SSE5
VPHSUBDQ	Xmmreg，xmmrm128 *	AMD，SSE5
VPHSUBWD	xmmreg,xmmrm128*	AMD,SSE5
VPHSUBWD	Xmmreg，xmmrm128 *	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSDD	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSDD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSDQH	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSDQH	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSDQL	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSDQL	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSSDD	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSSDD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSSDQH	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSSDQH	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSSDQL	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSSDQL	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSSWD	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSSWD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSSWW	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSSWW	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSWD	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSWD	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMACSWW	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMACSWW	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPMADCSSWD	Xmmreg，xmmreg *， xmmrm128， xmmreg	
VPMADCSSWD	AMD，SSE5	
VPMADCSWD	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	

VPMADCSWD	Xmmreg，xmmreg *，xmmrm128，xmmreg	
	AMD，SSE5	
	xmmreg,xmmreg*,xmmreg,xmmrm128 AMD,SSE5	
VPPERM	Xmmreg，xmmreg *，xmmreg，xmmrm128	
VPPERM	AMD，SSE5	
	xmmreg,xmmreg*,xmmrm128,xmmreg AMD,SSE5	
VPPERM	Xmmreg，xmmreg *，xmmrm128，xmmreg	
VPPERM	AMD，SSE5	
	xmmreg,xmmrm128*,xmmreg	
VPROTB	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTB	xmmreg	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128	
VPROTB	Xmmreg，xmmreg *，	AMD,SSE5
VPROTB	xmmrm128	AMD，SSE5
	xmmreg,xmmrm128*,imm8	
VPROTB	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTB	imm8	AMD，SSE5
	xmmreg,xmmrm128*,xmmreg	
VPROTD	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTD	xmmreg	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128	
VPROTD	Xmmreg，xmmreg *，	AMD,SSE5
VPROTD	xmmrm128	AMD，SSE5
	xmmreg,xmmrm128*,imm8	
VPROTD	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTD	imm8	AMD，SSE5
	xmmreg,xmmrm128*,xmmreg	
VPROTQ	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTQ	xmmreg	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128	
VPROTQ	Xmmreg，xmmreg *，	AMD,SSE5
VPROTQ	xmmrm128	AMD，SSE5
	xmmreg,xmmrm128*,imm8	
VPROTQ	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTQ	imm8	AMD，SSE5
	xmmreg,xmmrm128*,xmmreg	
VPROTW	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTW	xmmreg	AMD，SSE5
	xmmreg,xmmreg*,xmmrm128	
VPROTW	Xmmreg，xmmreg *，	AMD,SSE5
VPROTW	xmmrm128	AMD，SSE5
	xmmreg,xmmrm128*,imm8	
VPROTW	Xmmreg，xmmrm128 *，	AMD,SSE5
VPROTW	imm8	AMD，SSE5
	xmmreg,xmmrm128*,xmmreg	
VPSHAB	Xmmreg，xmmrm128 *，	AMD,SSE5
VPSHAB	xmmreg	AMD，SSE5

194

	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHAB	Xmmreg，xmmreg *，	AMD，
VPSHAB	xmmrm128	SSE5
VPSHAD	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHAD	Xmmreg，xmmrm128 *，	AMD，
VPSHAD	xmmreg	SSE5
VPSHAD	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHAD	Xmmreg，xmmreg *，	AMD，
VPSHAD	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHAQ	Xmmreg，xmmrm128 *，	AMD，
VPSHAQ	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHAQ	Xmmreg，xmmreg *，	AMD，
VPSHAQ	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHAW	Xmmreg，xmmrm128 *，	AMD，
VPSHAW	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHAW	Xmmreg，xmmreg *，	AMD，
VPSHAW	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHLB	Xmmreg，xmmrm128 *，	AMD，
VPSHLB	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHLB	Xmmreg，xmmreg *，	AMD，
VPSHLB	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHLD	Xmmreg，xmmrm128 *，	AMD，
VPSHLD	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHLD	Xmmreg，xmmreg *，	AMD，
VPSHLD	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHLQ	Xmmreg，xmmrm128 *，	AMD，
VPSHLQ	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHLQ	Xmmreg，xmmreg *，	AMD，
VPSHLQ	xmmrm128	SSE5
	xmmreg,xmmrm128*,xmmreg	AMD,SSE5
VPSHLW	Xmmreg，xmmrm128 *，	AMD，
VPSHLW	xmmreg	SSE5
	xmmreg,xmmreg*,xmmrm128	AMD,SSE5
VPSHLW	Xmmreg，xmmreg *，	AMD，
VPSHLW	xmmrm128	SSE5

B.1.34 Intel AVX2 instructions
B. 1.34 英特尔 avx2 指令

VMPSADBW	ymmreg,ymmreg*,ymmrm256,imm8 AVX2
VMPSADBW	Ymmreg，ymmreg *， ymmrm256，imm8

VMPSADBW	AVX2	
VPABSB	ymmreg,ymmrm256	AVX2
VPABSB	Ymmreg，ymm256	AVX2
VPABSW	ymmreg,ymmrm256	AVX2
VPABSW	Ymmreg，ymm256	AVX2
VPABSD	ymmreg,ymmrm256	AVX2
VPABSD	Ymmreg，ymm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPACKSSWB	Ymmreg，ymmreg * ，	AVX2
VPACKSSWB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPACKSSDW	Ymmreg，ymmreg * ，	AVX2
VPACKSSDW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPACKUSDW	Ymmreg，ymmreg * ，	AVX2
VPACKUSDW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPACKUSWB	Ymmreg，ymmreg * ，	AVX2
VPACKUSWB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDB	Ymmreg，ymmreg * ，	AVX2
VPADDB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDW	Ymmreg，ymmreg * ，	AVX2
VPADDW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDD	Ymmreg，ymmreg * ，	AVX2
VPADDD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDQ	Ymmreg，ymmreg * ，	AVX2
VPADDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDSB	Ymmreg，ymmreg * ，	AVX2
VPADDSB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDSW	Ymmreg，ymmreg * ，	AVX2
VPADDSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDUSB	Ymmreg，ymmreg * ，	AVX2
VPADDUSB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPADDUSW	Ymmreg，ymmreg * ，	AVX2
VPADDUSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256,imm8 AVX2	
VPALIGNR	Ymmreg，ymmreg * ， ymmrm256， imm8	
VPALIGNR	AVX2	
	ymmreg,ymmreg*,ymmrm256	
VPAND	Ymmreg，ymmreg * ，	AVX2
VPAND	ymmrm256	AVX2
VPANDN	ymmreg,ymmreg*,ymmrm256	AVX2
VPANDN	Ymmreg，ymmreg * ，	AVX2

	ymmrm256	
	ymmreg,ymmreg*,ymmrm256	
VPAVGB	Ymmreg， ymmreg *，	AVX2
VPAVGB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPAVGW	Ymmreg， ymmreg *，	AVX2
VPAVGW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256,ymmreg AVX2	
VPBLENDVB	Ymmreg， ymmreg *， ymmrm256， ymmreg	
VPBLENDVB	AVX2	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX2	
VPBLENDW	Ymmreg， ymmreg *， ymmrm256， imm8	
翻译	AVX2	
	ymmreg,ymmreg*,ymmrm256	
VPCMPEQB	Ymmreg， ymmreg *，	AVX2
VPCMPEQB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPEQW	Ymmreg， ymmreg *，	AVX2
VPCMPEQW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPEQD	Ymmreg， ymmreg *，	AVX2
VPCMPEQD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPEQQ	Ymmreg， ymmreg *，	AVX2
VPCMPEQQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPGTB	Ymmreg， ymmreg *，	AVX2
VPCMPGTB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPGTW	Ymmreg， ymmreg *，	AVX2
VPCMPGTW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPGTD	Ymmreg， ymmreg *，	AVX2
VPCMPGTD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPCMPGTQ	Ymmreg， ymmreg *，	AVX2
VPCMPGTQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPHADDW	Ymmreg， ymmreg *，	AVX2
VPHADDW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPHADDD	Ymmreg， ymmreg *，	AVX2
VPHADDD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPHADDSW	Ymmreg， ymmreg *，	AVX2
VPHADDSW	ymmrm256	AVX2
VPHSUBW	ymmreg,ymmreg*,ymmrm256	
VPHSUBW	Ymmreg， ymmreg *，	AVX2
VPHSUBW	ymmrm256	AVX2
VPHSUBD	ymmreg,ymmreg*,ymmrm256	AVX2
VPHSUBD	Ymmreg， ymmreg *，	AVX2

ymmrm256

195

	ymmreg,ymmreg*,ymmrm256	
VPHSUBSW	Ymmreg，ymmreg *，	AVX2
VPHSUBSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMADDUBSW	Ymmreg，ymmreg *，	AVX2
VPMADDUBSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMADDWD	Ymmreg，ymmreg *，	AVX2
VPMADDWD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXSB	Ymmreg，ymmreg *，	AVX2
VPMAXSB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXSW	Ymmreg，ymmreg *，	AVX2
VPMAXSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXSD	Ymmreg，ymmreg *，	AVX2
VPMAXSD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXUB	Ymmreg，ymmreg *，	AVX2
VPMAXUB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXUW	Ymmreg，ymmreg *，	AVX2
VPMAXUW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMAXUD	Ymmreg，ymmreg *，	AVX2
VPMAXUD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINSB	Ymmreg，ymmreg *，	AVX2
VPMINSB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINSW	Ymmreg，ymmreg *，	AVX2
VPMINSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINSD	Ymmreg，ymmreg *，	AVX2
VPMINSD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINUB	Ymmreg，ymmreg *，	AVX2
VPMINUB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINUW	Ymmreg，ymmreg *，	AVX2
VPMINUW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMINUD	Ymmreg，ymmreg *，	AVX2
VPMINUD	ymmrm256	AVX2
VPMOVMSKB	reg32,ymmreg	AVX2
VPMOVMSKB	32 岁，年轻人	AVX2
VPMOVMSKB	reg64,ymmreg	AVX2
VPMOVMSKB	Reg64，ymmreg	AVX2
VPMOVSXBW	ymmreg,xmmrm128	AVX2

VPMOVSXBW	Ymmreg，xmmrm128	AVX2
VPMOVSXBD	ymmreg,mem64	AVX2
VPMOVSXBD	Ymmreg，mem64	AVX2
VPMOVSXBD	ymmreg,xmmreg	AVX2
VPMOVSXBD	Ymmreg，xmreg	AVX2
VPMOVSXBQ	ymmreg,mem32	AVX2
VPMOVSXBQ	Ymmreg，mem32	AVX2
VPMOVSXBQ	ymmreg,xmmreg	AVX2
VPMOVSXBQ	Ymmreg，xmreg	AVX2
VPMOVSXWD	ymmreg,xmmrm128	AVX2
VPMOVSXWD	Ymmreg，xmmrm128	AVX2
VPMOVSXWQ	ymmreg,mem64	AVX2
VPMOVSXWQ	Ymmreg，mem64	AVX2
VPMOVSXWQ	ymmreg,xmmreg	AVX2
VPMOVSXWQ	Ymmreg，xmreg	AVX2
VPMOVSXDQ	ymmreg,xmmrm128	AVX2
VPMOVSXDQ	Ymmreg，xmmrm128	AVX2
VPMOVZXBW	ymmreg,xmmrm128	AVX2
VPMOVZXBW	Ymmreg，xmmrm128	AVX2
VPMOVZXBD	ymmreg,mem64	AVX2
VPMOVZXBD	Ymmreg，mem64	AVX2
VPMOVZXBD	ymmreg,xmmreg	AVX2
VPMOVZXBD	Ymmreg，xmreg	AVX2
VPMOVZXBQ	ymmreg,mem32	AVX2
VPMOVZXBQ	Ymmreg，mem32	AVX2
VPMOVZXBQ	ymmreg,xmmreg	AVX2
VPMOVZXBQ	Ymmreg，xmreg	AVX2
VPMOVZXWD	ymmreg,xmmrm128	AVX2
VPMOVZXWD	Ymmreg，xmmrm128	AVX2
VPMOVZXWQ	ymmreg,mem64	AVX2
VPMOVZXWQ	Ymmreg，mem64	AVX2
VPMOVZXWQ	ymmreg,xmmreg	AVX2
VPMOVZXWQ	Ymmreg，xmreg	AVX2
VPMOVZXDQ	ymmreg,xmmrm128	AVX2
VPMOVZXDQ	Ymmreg，xmmrm128	AVX2
VPMULDQ	ymmreg,ymmreg*,ymmrm256	AVX2
VPMULDQ	Ymmreg，ymmreg *，ymmrm256	AVX2
VPMULHRSW	ymmreg,ymmreg*,ymmrm256	AVX2
VPMULHRSW	Ymmreg，ymmreg *，ymmrm256	AVX2
VPMULHUW	ymmreg,ymmreg*,ymmrm256	AVX2
VPMULHUW	Ymmreg，ymmreg *，ymmrm256	AVX2
VPMULHW	ymmreg,ymmreg*,ymmrm256	AVX2
VPMULHW	Ymmreg，ymmreg *，ymmrm256	AVX2
VPMULLW	ymmreg,ymmreg*,ymmrm256	AVX2
Vmpmullw	Ymmreg，ymmreg *，ymmrm256	AVX2

	ymmreg,ymmreg*,ymmrm256	
VPMULLD	Ymmreg，ymmreg *，	AVX2
VPMULLD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPMULUDQ	Ymmreg，ymmreg *，	AVX2
VPMULUDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPOR	Ymmreg，ymmreg *，	AVX2
VPOR	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSADBW	Ymmreg，ymmreg *，	AVX2
VPSADBW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSHUFB	Ymmreg，ymmreg *，	AVX2
VPSHUFB	ymmrm256	AVX2
VPSHUFD	ymmreg,ymmrm256,imm8	AVX2
VPSHUFD	Ymmreg，ymmrm256，imm8	AVX2
VPSHUFHW	ymmreg,ymmrm256,imm8	AVX2
这是什么意思	Ymmreg，ymmrm256，imm8	AVX2
VPSHUFLW	ymmreg,ymmrm256,imm8	AVX2
(翻译)	Ymmreg，ymmrm256，imm8	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSIGNB	Ymmreg，ymmreg *，	AVX2
VPSIGNB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSIGNW	Ymmreg，ymmreg *，	AVX2
VPSIGNW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSIGND	Ymmreg，ymmreg *，	AVX2
VPSIGND	ymmrm256	AVX2
VPSLLDQ	ymmreg,ymmreg*,imm8	AVX2
VPSLLDQ	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSLLW	Ymmreg，ymmreg *，	AVX2
VPSLLW	xmmrm128	AVX2
VPSLLW	ymmreg,ymmreg*,imm8	AVX2
VPSLLW	Ymmreg *， imm8	AVX2

196

	ymmreg,ymmreg*,xmmrm128	
VPSLLD	Ymmreg， ymmreg *，	AVX2
VPSLLD	xmmrm128	AVX2
VPSLLD	ymmreg,ymmreg*,imm8	AVX2
VPSLLD	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSLLQ	Ymmreg， ymmreg *，	AVX2
VPSLLQ	xmmrm128	AVX2
VPSLLQ	ymmreg,ymmreg*,imm8	AVX2
VPSLLQ	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSRAW	Ymmreg， ymmreg *，	AVX2
VPSRAW	xmmrm128	AVX2
VPSRAW	ymmreg,ymmreg*,imm8	AVX2
VPSRAW	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSRAD	Ymmreg， ymmreg *，	AVX2
VPSRAD	xmmrm128	AVX2
VPSRAD	ymmreg,ymmreg*,imm8	AVX2
VPSRAD	Ymmreg *， imm8	AVX2
VPSRLDQ	ymmreg,ymmreg*,imm8	AVX2
VPSRLDQ	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSRLW	Ymmreg， ymmreg *，	AVX2
VPSRLW	xmmrm128	AVX2
VPSRLW	ymmreg,ymmreg*,imm8	AVX2
VPSRLW	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSRLD	Ymmreg， ymmreg *，	AVX2
VPSRLD	xmmrm128	AVX2
VPSRLD	ymmreg,ymmreg*,imm8	AVX2
VPSRLD	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,xmmrm128	
VPSRLQ	Ymmreg， ymmreg *，	AVX2
VPSRLQ	xmmrm128	AVX2
VPSRLQ	ymmreg,ymmreg*,imm8	AVX2
VPSRLQ	Ymmreg *， imm8	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBB	Ymmreg， ymmreg *，	AVX2
VPSUBB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBW	Ymmreg， ymmreg *，	AVX2
VPSUBW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBD	Ymmreg， ymmreg *，	AVX2
VPSUBD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBQ	Ymmreg， ymmreg *，	AVX2
VPSUBQ	ymmrm256	AVX2
VPSUBSB	ymmreg,ymmreg*,ymmrm256	AVX2

VPSUBSB	Ymmreg，ymmreg *，	AVX2
	ymmrm256	
	ymmreg,ymmreg*,ymmrm256	
VPSUBSW	Ymmreg，ymmreg *，	AVX2
VPSUBSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBUSB	Ymmreg，ymmreg *，	AVX2
VPSUBUSB	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSUBUSW	Ymmreg，ymmreg *，	AVX2
VPSUBUSW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKHBW	Ymmreg，ymmreg *，	AVX2
VPUNPCKHBW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKHWD	Ymmreg，ymmreg *，	AVX2
VPUNPCKHWD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKHDQ	Ymmreg，ymmreg *，	AVX2
VPUNPCKHDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKHQDQ	Ymmreg，ymmreg *，	AVX2
VPUNPCKHQDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKLBW	Ymmreg，ymmreg *，	AVX2
VPUNPCKLBW	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKLWD	Ymmreg，ymmreg *，	AVX2
VPUNPCKLWD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKLDQ	Ymmreg，ymmreg *，	AVX2
VPUNPCKLDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPUNPCKLQDQ	Ymmreg，ymmreg *，	AVX2
VPUNPCKLQDQ	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPXOR	Ymmreg，ymmreg *，	AVX2
VPXOR	ymmrm256	AVX2
VMOVNTDQA	ymmreg,mem256	AVX2
VMOVNTDQA	Ymmreg，mem256	AVX2
VBROADCASTSS		
Vbroadcasts.vbroad	xmmreg,xmmreg	AVX2
cast	Xmmreg，xmmreg	AVX2
VBROADCASTSS		
Vbroadcasts.vbroad	ymmreg,xmmreg	AVX2
cast	Ymmreg，xmreg	AVX2
VBROADCASTSD	ymmreg,xmmreg	AVX2
VBROADCASTSD	Ymmreg，xmreg	AVX2
VBROADCASTI128	ymmreg,mem128	AVX2
VBROADCASTI128	Ymmreg mem128	AVX2
VPBLENDD	xmmreg,xmmreg*,xmmrm128,imm8	AVX2

VPBLENDD	Xmmreg，xmmreg *，xmmrm128，imm8 AVX2	
	ymmreg,ymmreg*,ymmrm256,imm8 AVX2	
VPBLENDD	Ymmreg，ymmreg *，ymmrm256，imm8	
VPBLENDD	AVX2	
VPBROADCASTB	xmmreg,mem8	AVX2
VPBROADCASTB	Xmmreg，mem8	AVX2
VPBROADCASTB	xmmreg,xmmreg	AVX2
VPBROADCASTB	Xmmreg，xmmreg	AVX2
VPBROADCASTB	ymmreg,mem8	AVX2
VPBROADCASTB	Ymmreg，mem8	AVX2
VPBROADCASTB	ymmreg,xmmreg	AVX2
VPBROADCASTB	Ymmreg，xmreg	AVX2
VPBROADCASTW	xmmreg,mem16	AVX2
VPBROADCASTW	Xmmreg，mem16	AVX2
VPBROADCASTW	xmmreg,xmmreg	AVX2
VPBROADCASTW	Xmmreg，xmmreg	AVX2
VPBROADCASTW	ymmreg,mem16	AVX2
VPBROADCASTW	Ymmreg，mem16	AVX2
VPBROADCASTW	ymmreg,xmmreg	AVX2
VPBROADCASTW	Ymmreg，xmreg	AVX2
VPBROADCASTD	xmmreg,mem32	AVX2
VPBROADCASTD	Xmmreg，mem32	AVX2
VPBROADCASTD	xmmreg,xmmreg	AVX2
VPBROADCASTD	Xmmreg，xmmreg	AVX2
VPBROADCASTD	ymmreg,mem32	AVX2
VPBROADCASTD	Ymmreg，mem32	AVX2
VPBROADCASTD	ymmreg,xmmreg	AVX2
VPBROADCASTD	Ymmreg，xmreg	AVX2
VPBROADCASTQ	xmmreg,mem64	AVX2
VPBROADCASTQ	Xmmreg，mem64	AVX2
VPBROADCASTQ	xmmreg,xmmreg	AVX2
VPBROADCASTQ	Xmmreg，xmmreg	AVX2
VPBROADCASTQ	ymmreg,mem64	AVX2
VPBROADCASTQ	Ymmreg，mem64	AVX2

197

VPBROADCASTQ	ymmreg,xmmreg	AVX2
VPBROADCASTQ	Ymmreg，xmreg	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPERMD	Ymmreg，ymmreg＊，	AVX2
VPERMD	ymmrm256	AVX2
VPERMPD	ymmreg,ymmrm256,imm8	AVX2
VPERMPD	Ymmreg，ymmrm256，imm8	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPERMPS	Ymmreg，ymmreg＊，	AVX2
VPERMPS	ymmrm256	AVX2
VPERMQ	ymmreg,ymmrm256,imm8	AVX2
VPERMQ	Ymmreg，ymmrm256，imm8	AVX2
	ymmreg,ymmreg*,ymmrm256,imm8 AVX2	
VPERM2I128	Ymmreg，ymmreg＊，ymmrm256，imm8	
VPERM2I128	AVX2	
VEXTRACTI128	xmmrm128,ymmreg,imm8	AVX2
VEXTRACTI128	Xmmrm128，ymmreg，imm8	AVX2
	ymmreg,ymmreg*,xmmrm128,imm8 AVX2	
VINSERTI128	Ymmreg，ymmreg＊，xmmrm128，imm8	
VINSERTI128	AVX2	
	xmmreg,xmmreg*,mem128	
VPMASKMOVD	Xmmreg，xmmreg＊，	AVX2
VPMASKMOVD	mem128	AVX2
VPMASKMOVD	ymmreg,ymmreg*,mem256	AVX2
VPMASKMOVD	Ymmreg＊mem256	AVX2
	xmmreg,xmmreg*,mem128	
VPMASKMOVQ	Xmmreg，xmmreg＊，	AVX2
VPMASKMOVQ	mem128	AVX2
VPMASKMOVQ	ymmreg,ymmreg*,mem256	AVX2
VPMASKMOVQ	Ymmreg＊mem256	AVX2
	mem128,xmmreg*,xmmreg	
VPMASKMOVD	Mem128，xmmreg＊，	AVX2
VPMASKMOVD	xmmreg	AVX2
	mem256,ymmreg*,ymmreg	
VPMASKMOVD	Mem256，ymmreg＊，	AVX2
VPMASKMOVD	ymmreg	AVX2
	mem128,xmmreg*,xmmreg	
VPMASKMOVQ	Mem128，xmmreg＊，	AVX2
VPMASKMOVQ	xmmreg	AVX2
	mem256,ymmreg*,ymmreg	
VPMASKMOVQ	Mem256，ymmreg＊，	AVX2
VPMASKMOVQ	ymmreg	AVX2
	xmmreg,xmmreg*,xmmrm128	
VPSLLVD	Xmmreg，xmmreg＊，	AVX2
VPSLLVD	xmmrm128	AVX2
	xmmreg,xmmreg*,xmmrm128	
VPSLLVQ	Xmmreg，xmmreg＊，	AVX2
VPSLLVQ	xmmrm128	AVX2
VPSLLVD	ymmreg,ymmreg*,ymmrm256	AVX2
VPSLLVD	Ymmreg，ymmreg＊，	AVX2

	ymmrm256	
	ymmreg,ymmreg*,ymmrm256	
VPSLLVQ	Ymmreg，ymmreg * ，	AVX2
VPSLLVQ	ymmrm256	AVX2
	xmmreg,xmmreg*,xmmrm128	
VPSRAVD	Xmmreg，xmmreg * ，	AVX2
VPSRAVD	xmmrm128	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSRAVD	Ymmreg，ymmreg * ，	AVX2
VPSRAVD	ymmrm256	AVX2
	xmmreg,xmmreg*,xmmrm128	
VPSRLVD	Xmmreg，xmmreg * ，	AVX2
VPSRLVD	xmmrm128	AVX2
	xmmreg,xmmreg*,xmmrm128	
VPSRLVQ	Xmmreg，xmmreg * ，	AVX2
VPSRLVQ	xmmrm128	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSRLVD	Ymmreg，ymmreg * ，	AVX2
VPSRLVD	ymmrm256	AVX2
	ymmreg,ymmreg*,ymmrm256	
VPSRLVQ	Ymmreg，ymmreg * ，	AVX2
VPSRLVQ	ymmrm256	AVX2
VGATHERDPD	xmmreg,xmem64,xmmreg	AVX2
Vgather dpd	Xmmreg，xmem64，xmmreg	AVX2
VGATHERQPD	xmmreg,xmem64,xmmreg	AVX2
Vgather qpd	Xmmreg，xmem64，xmmreg	AVX2
VGATHERDPD	ymmreg,xmem64,ymmreg	AVX2
Vgather dpd	Ymmreg，xmem64，ymmreg	AVX2
VGATHERQPD	ymmreg,ymem64,ymmreg	AVX2
Vgather qpd	Ymmreg ymem64 ymmreg	AVX2
VGATHERDPS	xmmreg,xmem32,xmmreg	AVX2
Vgather dps	Xmmreg，xmem32，xmmreg	AVX2
VGATHERQPS	xmmreg,xmem32,xmmreg	AVX2
Vgather qps	Xmmreg，xmem32，xmmreg	AVX2
VGATHERDPS	ymmreg,ymem32,ymmreg	AVX2
Vgather dps	Ymmreg，ymem32，ymmreg	AVX2
VGATHERQPS	xmmreg,ymem32,xmmreg	AVX2
Vgather qps	Xmmreg，ymem32，xmmreg	AVX2
VPGATHERDD	xmmreg,xmem32,xmmreg	AVX2
Vpgather dd	Xmmreg，xmem32，xmmreg	AVX2
VPGATHERQD	xmmreg,xmem32,xmmreg	AVX2
Vpgather qd	Xmmreg，xmem32，xmmreg	AVX2
VPGATHERDD	ymmreg,ymem32,ymmreg	AVX2
Vpgather dd	Ymmreg，ymem32，ymmreg	AVX2
VPGATHERQD	xmmreg,ymem32,xmmreg	AVX2
Vpgather qd	Xmmreg，ymem32，xmmreg	AVX2
VPGATHERDQ	xmmreg,xmem64,xmmreg	AVX2
Vpgather dq	Xmmreg，xmem64，xmmreg	AVX2
VPGATHERQQ	xmmreg,xmem64,xmmreg	AVX2
Vpgather qq	Xmmreg，xmem64，xmmreg	AVX2
VPGATHERDQ	ymmreg,xmem64,ymmreg	AVX2

Vpgather dq	Ymmreg，xmem64，ymmreg	AVX2
VPGATHERQQ	ymmreg,ymem64,ymmreg	AVX2
Vpgather qq	Ymmreg ymem64 ymmreg	AVX2

B.1.35 Intel Transactional Synchronization Extensions (TSX)
B. 1.35 英特尔事务同步扩展(TSX)

XABORT	imm	RTM
XABORT	是的	RTM
XABORT	imm8	RTM
XABORT	Im8	RTM
XBEGIN	imm	RTM
XBEGIN	是的	RTM
XBEGIN	imm|near	RTM,ND
XBEGIN	在附近	RTM，ND
XBEGIN	imm16	RTM,NOLONG
XBEGIN	Im16	RTM，NOLONG
		RTM,NOLONG,ND
XBEGIN	imm16|near	RTM，
XBEGIN	Im16 | 近	NOLONG，ND
XBEGIN	imm32	RTM,NOLONG
XBEGIN	Im32	RTM，NOLONG
		RTM,NOLONG,ND
XBEGIN	imm32|near	RTM，
XBEGIN	Im32 | 近	NOLONG，ND
XBEGIN	imm64	RTM,LONG
XBEGIN	Im64	RTM，LONG

198

```
XBEGIN           imm64|near                RTM,LONG,ND
XBEGIN imm64 | 靠近 RTM, LONG, ND
XEND                                       RTM
Xendrtm
XTEST                                      HLE,RTM
XTEST HLE, RTM
```

B.1.36 Intel BMI1 and BMI2 instructions, AMD TBM instructions
英特尔 bmi1 和 bmi2 指令，AMD TBM 指令

ANDN	reg32,reg32,rm32	BMI1
ANDN	Reg32，reg32，rm32	BMI1
		LONG,BMI1
ANDN	reg64,reg64,rm64	LONG，
ANDN	Reg64，reg64，rm64	BMI1
BEXTR	reg32,rm32,reg32	BMI1
BEXTR	Reg32，rm32，reg32	BMI1
		LONG,BMI1
BEXTR	reg64,rm64,reg64	LONG，
BEXTR	Reg64，rm64，reg64	BMI1
BEXTR	reg32,rm32,imm32	TBM
BEXTR	Reg32，rm32，imm32	TBM
		LONG,TBM
		LONG，
BEXTR	reg64,rm64,imm32	TBM 龙，
BEXTR	Reg64，rm64，imm32	TBM
BLCI	reg32,rm32	TBM
BLCI	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLCI	reg64,rm64	TBM 龙，
BLCI	Reg64，rm64	TBM
BLCIC	reg32,rm32	TBM
BLCIC	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLCIC	reg64,rm64	TBM 龙，
BLCIC	Reg64，rm64	TBM
BLSI	reg32,rm32	BMI1
BLSI	Reg32，rm32	BMI1
		LONG,BMI1
BLSI	reg64,rm64	LONG，
BLSI	Reg64，rm64	BMI1
BLSIC	reg32,rm32	TBM
BLSIC	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLSIC	reg64,rm64	TBM 龙，
BLSIC	Reg64，rm64	TBM
BLCFILL	reg32,rm32	TBM
BLCFILL	Reg32，rm32	TBM
BLCFILL	reg64,rm64	LONG,TBM

BLCFILL	Reg64，rm64	LONG，
		TBM 龙，
		TBM
BLSFILL	reg32,rm32	TBM
BLSFILL	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLSFILL	reg64,rm64	TBM 龙，
BLSFILL	Reg64，rm64	TBM
BLCMSK	reg32,rm32	TBM
BLCMSK	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLCMSK	reg64,rm64	TBM 龙，
BLCMSK	Reg64，rm64	TBM
BLSMSK	reg32,rm32	BMI1
BLSMSK	Reg32，rm32	BMI1
		LONG,BMI1
BLSMSK	reg64,rm64	LONG，
BLSMSK	Reg64，rm64	BMI1
BLSR	reg32,rm32	BMI1
BLSR	Reg32，rm32	BMI1
		LONG,BMI1
BLSR	reg64,rm64	LONG，
BLSR	Reg64，rm64	BMI1
BLCS	reg32,rm32	TBM
BLCS	Reg32，rm32	TBM
		LONG,TBM
		LONG，
BLCS	reg64,rm64	TBM 龙，
BLCS	Reg64，rm64	TBM
BZHI	reg32,rm32,reg32	BMI2
BZHI	Reg32，rm32，reg32	BMI2
BZHI	reg64,rm64,reg64	LONG,BMI2
BZHI	Reg64，rm64，reg64	LONG BMI2
MULX	reg32,reg32,rm32	BMI2
MULX	Reg32，reg32，rm32	BMI2
MULX	reg64,reg64,rm64	LONG,BMI2
MULX	Reg64，reg64，rm64	LONG BMI2
PDEP	reg32,reg32,rm32	BMI2
PDEP	Reg32，reg32，rm32	BMI2
PDEP	reg64,reg64,rm64	LONG,BMI2
PDEP	Reg64，reg64，rm64	LONG BMI2
PEXT	reg32,reg32,rm32	BMI2
PEXT	Reg32，reg32，rm32	BMI2
PEXT	reg64,reg64,rm64	LONG,BMI2
PEXT	Reg64，reg64，rm64	LONG BMI2
RORX	reg32,rm32,imm8	BMI2
RORX	Reg32，rm32，imm8	BMI2
RORX	reg64,rm64,imm8	LONG,BMI2

RORX	Reg64，rm64，imm8	LONG BMI2
SARX	reg32,rm32,reg32	BMI2
SARX	Reg32，rm32，reg32	BMI2
SARX	reg64,rm64,reg64	LONG,BMI2
SARX	Reg64，rm64，reg64	LONG BMI2
SHLX	reg32,rm32,reg32	BMI2
SHLX	Reg32，rm32，reg32	BMI2
SHLX	reg64,rm64,reg64	LONG,BMI2
SHLX	Reg64，rm64，reg64	LONG BMI2
SHRX	reg32,rm32,reg32	BMI2
SHRX	Reg32，rm32，reg32	BMI2
SHRX	reg64,rm64,reg64	LONG,BMI2
SHRX	Reg64，rm64，reg64	LONG BMI2
TZCNT	reg16,rm16	BMI1
TZCNT	正版 16，rm16	BMI1
TZCNT	reg32,rm32	BMI1
TZCNT	Reg32，rm32	BMI1
		LONG,BMI1
TZCNT	reg64,rm64	LONG，
TZCNT	Reg64，rm64	BMI1
TZMSK	reg32,rm32	TBM
TZMSK	Reg32，rm32	TBM
		LONG,TBM
		LONG，
TZMSK	reg64,rm64	TBM 龙，
TZMSK	Reg64，rm64	TBM
T1MSKC	reg32,rm32	TBM
T1MSKC	Reg32，rm32	TBM

```
T1MSKC           reg64,rm64               LONG,TBM
T1MSKC reg64， rm64 LONG， TBM
PREFETCHWT1      mem8                     PREFETCHWT1
PREFETCHWT1 mem8 PREFETCHWT1
```

B.1.37 Intel Memory Protection Extensions (MPX)
B. 1.37 英特尔内存保护扩展插件(MPX)

BNDMK	bndreg,mem	MPX,MIB
BNDMK	Bndreg， mem	MPX MIB
BNDCL	bndreg,mem	MPX
BNDCL	Bndreg， mem	MPX
		MPX,NOLONG
BNDCL	bndreg,reg32	MPX， NOLONG
BNDCL	Bndreg， reg32	MPX， NOLONG
BNDCL	bndreg,reg64	MPX,LONG
BNDCL	Bndreg， reg64	MPX， LONG
BNDCU	bndreg,mem	MPX
BNDCU	Bndreg， mem	MPX
		MPX,NOLONG
BNDCU	bndreg,reg32	MPX， NOLONG
BNDCU	Bndreg， reg32	MPX， NOLONG
BNDCU	bndreg,reg64	MPX,LONG
BNDCU	Bndreg， reg64	MPX， LONG
BNDCN	bndreg,mem	MPX
BNDCN	Bndreg， mem	MPX
		MPX,NOLONG
BNDCN	bndreg,reg32	MPX， NOLONG
BNDCN	Bndreg， reg32	MPX， NOLONG
BNDCN	bndreg,reg64	MPX,LONG
BNDCN	Bndreg， reg64	MPX， LONG
BNDMOV	bndreg,bndreg	MPX
BNDMOV	Bndreg， bndreg	MPX
BNDMOV	bndreg,mem	MPX
BNDMOV	Bndreg， mem	MPX
BNDMOV	bndreg,bndreg	MPX
BNDMOV	Bndreg， bndreg	MPX
BNDMOV	mem,bndreg	MPX
BNDMOV	Mem， bndreg	MPX
BNDLDX	bndreg,mem	MPX,MIB
BNDLDX	Bndreg， mem	MPX MIB
		MPX,MIB,NOLON
		G
		MPX， MIB，
BNDLDX	bndreg,mem,reg32	NOLONG MPX，
BNDLDX	Bndreg， mem， reg32	MIB， NOLONG
BNDLDX	bndreg,mem,reg64	MPX,MIB,LONG
BNDLDX	Bndreg， mem， reg64	MPX MIB LONG
BNDSTX	mem,bndreg	MPX,MIB
BNDSTX	Mem， bndreg	MPX MIB
BNDSTX	mem,reg32,bndreg	MPX,MIB,NOLON

BNDSTX	Mem，reg32，bndreg	G
		MPX，MIB，
		NOLONG MPX，
		MIB，NOLONG
BNDSTX	mem,reg64,bndreg	MPX,MIB,LONG
BNDSTX	Mem，reg64，bndreg	MPX MIB LONG
		MPX,MIB,NOLON
		G
		MPX，MIB，
BNDSTX	mem,bndreg,reg32	NOLONG MPX，
BNDSTX	Mem，bndreg，reg32	MIB，NOLONG
BNDSTX	mem,bndreg,reg64	MPX,MIB,LONG
BNDSTX	Mem，bndreg，reg64	MPX MIB LONG

B.1.38 Intel SHA acceleration instructions
1.38 英特尔 SHA 加速指令

		SH
		A
		民
		政
		事
		务
SHA1MSG1	xmmreg,xmmrm128	局
SHA1MSG1	Xmmreg，xmmrm128	长
		SH
		A
		民
		政
		事
		务
SHA1MSG2	xmmreg,xmmrm128	局
SHA1MSG2	Xmmreg，xmmrm128	长
		SH
		A
		民
		政
		事
		务
SHA1NEXTE	xmmreg,xmmrm128	局
SHA1NEXTE	Xmmreg，xmmrm128	长
		SH
		A
		民
		政
		事
	xmmreg,xmmrm128,imm8	务
SHA1RNDS4	Xmmreg，xmmmrm128，	局
SHA1RNDS4	imm8	长
SHA256MSG1	xmmreg,xmmrm128	SH
SHA256MSG1	Xmmreg，xmmrm128	A

民政事务局长SHA民政事务局长SHA民政事务局长SHA民政事务局长

SHA256MSG2 xmmreg,xmmrm128
SHA256MSG2 Xmmreg，xmmrm128

xmmreg,xmmrm128,xmm0
SHA256RNDS2 Xmmreg，xmmrm128，
SHA256RNDS2 xmm0

SHA256RNDS2 xmmreg,xmmrm128
SHA256RNDS2 Xmmreg，xmmrm128

B.1.39 AVX−512 mask register instructions
B. 1.39 AVX-512 掩模寄存器说明书

```
KADDB            kreg,kreg,kreg
KADDB kreg, kreg, kreg
KADDD            kreg,kreg,kreg
KADDD kreg, kreg, kreg
KADDQ            kreg,kreg,kreg
KADDQ kreg, kreg, kreg
KADDW            kreg,kreg,kreg
KADDW 克雷格，克雷格，克雷格
KANDB            kreg,kreg,kreg
KANDB kreg, kreg, kreg
KANDD            kreg,kreg,kreg
坎德克雷格，克雷格，克雷格
KANDNB           kreg,kreg,kreg
KANDNB kreg, kreg, kreg
KANDND           kreg,kreg,kreg
KANDND kreg, kreg, kreg
KANDNQ           kreg,kreg,kreg
KANDNQ kreg, kreg, kreg
```

```
KANDNW            kreg,kreg,kreg
KANDNW kreg, kreg, kreg
KANDQ             kreg,kreg,kreg
KANDQ kreg, kreg, kreg
KANDW             kreg,kreg,kreg
KANDW kreg, kreg, kreg
KMOVB             kreg,krm8
KMOVB kreg, krm8
KMOVB             mem8,kreg
KMOVB mem8, kreg
KMOVB             kreg,reg32
KMOVB kreg, reg32
```

200

```
KMOVB            reg32,kreg
KMOVB reg32, kreg
KMOVD            kreg,krm32
KMOVD kreg, krm32
KMOVD            mem32,kreg
KMOVD mem32, kreg
KMOVD            kreg,reg32
KMOVD kreg, reg32
KMOVD            reg32,kreg
KMOVD reg32, kreg
KMOVQ            kreg,krm64
KMOVQ kreg, krm64
KMOVQ            mem64,kreg
KMOVQ mem64, kreg
KMOVQ            kreg,reg64
KMOVQ kreg, reg64
KMOVQ            reg64,kreg
KMOVQ reg64, kreg
KMOVW            kreg,krm16
KMOVW kreg, krm16
KMOVW            mem16,kreg
KMOVW mem16, kreg
KMOVW            kreg,reg32
KMOVW kreg, reg32
KMOVW            reg32,kreg
KMOVW reg32, kreg
KNOTB            kreg,kreg
KNOTB kreg, kreg
KNOTD            kreg,kreg
KNOTD kreg, kreg
KNOTQ            kreg,kreg
克雷格，克雷格
KNOTW            kreg,kreg
KNOTW kreg, kreg
KORB             kreg,kreg,kreg
KORB kreg, kreg, kreg
KORD             kreg,kreg,kreg
KORD kreg, kreg, kreg
KORQ             kreg,kreg,kreg
KORQ kreg, kreg, kreg
KORTESTB         kreg,kreg
KORTESTB kreg, kreg
KORTESTD         kreg,kreg
克雷格，克雷格
KORTESTQ         kreg,kreg
KORTESTQ kreg, kreg
KORTESTW         kreg,kreg
KORTESTW kreg, kreg
KORW             kreg,kreg,kreg
KORW kreg, kreg, kreg
KSHIFTLB         kreg,kreg,imm8
KSHIFTLB kreg, kreg, imm8
KSHIFTLD         kreg,kreg,imm8
KSHIFTLD kreg, kreg, imm8
KSHIFTLQ         kreg,kreg,imm8
KSHIFTLQ kreg, kreg, imm8
KSHIFTLW         kreg,kreg,imm8
KSHIFTLW kreg, kreg, imm8
KSHIFTRB         kreg,kreg,imm8
KSHIFTRB kreg, kreg, imm8
KSHIFTRD         kreg,kreg,imm8
KSHIFTRD kreg, kreg, imm8
KSHIFTRQ         kreg,kreg,imm8
KSHIFTRQ kreg, kreg, imm8
KSHIFTRW         kreg,kreg,imm8
KSHIFTRW kreg, kreg, imm8
KTESTB           kreg,kreg
克雷格，克雷格
KTESTD           kreg,kreg
KTESTD kreg, kreg
KTESTQ           kreg,kreg
KTESTQ kreg, kreg
KTESTW           kreg,kreg
KTESTW kreg, kreg
KUNPCKBW         kreg,kreg,kreg
克雷格，克雷格
KUNPCKDQ         kreg,kreg,kreg
KUNPCKDQ kreg, kreg, kreg
KUNPCKWD         kreg,kreg,kreg
KUNPCKWD 克雷格，克雷格，克雷格
KXNORB           kreg,kreg,kreg
KXNORB kreg, kreg, kreg
KXNORD           kreg,kreg,kreg
KXNORD 克雷格，克雷格，克雷格
KXNORQ           kreg,kreg,kreg
KXNORQ kreg, kreg, kreg
KXNORW           kreg,kreg,kreg
克雷格，克雷格，克雷格
KXORB            kreg,kreg,kreg
KXORB kreg, kreg, kreg
KXORD            kreg,kreg,kreg
KXORD 克雷格，克雷格，克雷格
KXORQ            kreg,kreg,kreg
KXORQ kreg, kreg, kreg
KXORW            kreg,kreg,kreg
KXORW kreg, kreg, kreg
```

B.1.40 AVX−512 instructions
B. 1.40 AVX-512 指令

```
                 xmmreg|mask|z,xmmreg*,xmmrm128|b64
VADDPD           Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VADDPD           | b64                                    AVX512VL
                 ymmreg|mask|z,ymmreg*,ymmrm256|b64
VADDPD           Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL
VADDPD           | b64                                    AVX512VL
                 zmmreg|mask|z,zmmreg*,zmmrm512|b64|er AVX512
VADDPD           Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | er
VADDPD           AVX512
```

201

```
VADDPS          xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VADDPS          Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VADDPS          ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VADDPS          Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
                zmmreg|mask|z,zmmreg*,zmmrm512|b32|er AVX512
VADDPS          Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | er
VADDPS          AVX512
VADDSD          xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512
VADDSD          Xmmreg | mask | z， xmmreg * ， xmmrm64 | er AVX512
VADDSS          xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512
VADDSS          Xmmreg | mask | z， xmmreg * ， xmmrm32 | er AVX512
                xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8 AVX512VL
VALIGND         Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32， imm8
VALIGND         AVX512VL
                ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL
VALIGND         Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32， imm8
VALIGND         AVX512VL
                zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8 AVX512
VALIGND         Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32， imm8
VALIGND         AVX512
                xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VL
VALIGNQ         Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64， imm8
VALIGNQ         AVX512VL
                ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL
VALIGNQ         Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64， imm8
VALIGNQ         AVX512VL
                zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512
VALIGNQ         Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64， imm8
VALIGNQ         AVX512
                xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/DQ
VANDNPD         Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VANDNPD         AVX512VL/DQ
                ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/DQ
VANDNPD         Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VANDNPD         AVX512VL/DQ
VANDNPD         zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512DQ
VANDNPD         Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512DQ
                xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/DQ
VANDNPS         Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VANDNPS         AVX512VL/DQ
                ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/DQ
VANDNPS         Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VANDNPS         AVX512VL/DQ
VANDNPS         zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512DQ
VANDNPS         Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512DQ
                xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/DQ
VANDPD          Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VANDPD          AVX512VL/DQ
VANDPD          ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/DQ
VANDPD          Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
```

	AVX512VL/DQ	
VANDPD	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512DQ	
VANDPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512DQ	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/DQ	
VANDPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32	
VANDPS	AVX512VL/DQ	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/DQ	
VANDPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32	
VANDPS	AVX512VL/DQ	
VANDPS	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512DQ	
VANDPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512DQ	
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VBLENDMPD	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VBLENDMPD	| b64	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VBLENDMPD	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VBLENDMPD	| b64	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VBLENDMPD	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VBLENDMPD	| b64	AVX512
	xmmreg|mask|z,xmmreg,xmmrm128|b32	
VBLENDMPS	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VBLENDMPS	| b32	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b32	
VBLENDMPS	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VBLENDMPS	| b32	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b32	
VBLENDMPS	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VBLENDMPS	| b32	AVX512
	ymmreg|mask|z,xmmrm64	
VBROADCASTF32X2	Ymmreg | mask | z，	AVX512VL/DQ
VBROADCASTF32X2	xmmrm64	AVX512VL/DQ
	zmmreg|mask|z,xmmrm64	
VBROADCASTF32X2	Zmmrreg | mask | z，	AVX512DQ
VBROADCASTF32X2	xmmrm64	AVX512DQ
VBROADCASTF32X4	ymmreg|mask|z,mem128	AVX512VL
VBROADCASTF32X4	Ymmreg | mask | z， mem128	AVX512VL
VBROADCASTF32X4	zmmreg|mask|z,mem128	AVX512
VBROADCASTF32X4	Zmmreg | mask | z， mem128	AVX512
VBROADCASTF32X8	zmmreg|mask|z,mem256	AVX512DQ
VBROADCASTF32X8	Zmmreg | mask | z， mem256	AVX512DQ
VBROADCASTF64X2	ymmreg|mask|z,mem128	AVX512VL/DQ
VBROADCASTF64X2	Ymmreg | mask | z， mem128	AVX512VL/DQ
VBROADCASTF64X2	zmmreg|mask|z,mem128	AVX512DQ
VBROADCASTF64X2	Zmmreg | mask | z， mem128	AVX512DQ
VBROADCASTF64X4	zmmreg|mask|z,mem256	AVX512
VBROADCASTF64X4	Zmmreg | mask | z， mem256	AVX512
	xmmreg|mask|z,xmmrm64	
VBROADCASTI32X2	Xmmreg | mask | z，	AVX512VL/DQ
VBROADCASTI32X2	xmmrm64	AVX512VL/DQ
VBROADCASTI32X2	ymmreg|mask|z,xmmrm64	AVX512VL/DQ

VBROADCASTI32X2	Ymmreg | mask | z，xmmrm64 zmmreg|mask|z,xmmrm64	AVX512VL/DQ
VBROADCASTI32X2	Zmmrreg | mask | z，	AVX512DQ
VBROADCASTI32X2	xmmrm64	AVX512DQ
VBROADCASTI32X4	ymmreg|mask|z,mem128	AVX512VL
VBROADCASTI32X4	Ymmreg | mask | z， mem128	AVX512VL
VBROADCASTI32X4	zmmreg|mask|z,mem128	AVX512
VBROADCASTI32X4	Zmmreg | mask | z， mem128	AVX512
VBROADCASTI32X8	zmmreg|mask|z,mem256	AVX512DQ
VBROADCASTI32X8	Zmmreg | mask | z， mem256	AVX512DQ
VBROADCASTI64X2	ymmreg|mask|z,mem128	AVX512VL/DQ
VBROADCASTI64X2	Ymmreg | mask | z， mem128	AVX512VL/DQ
VBROADCASTI64X2	zmmreg|mask|z,mem128	AVX512DQ
VBROADCASTI64X2	Zmmreg | mask | z， mem128	AVX512DQ
VBROADCASTI64X4	zmmreg|mask|z,mem256	AVX512
VBROADCASTI64X4	Zmmreg | mask | z， mem256	AVX512
VBROADCASTSD	ymmreg|mask|z,mem64	AVX512VL
VBROADCASTSD	Ymmreg | mask | z， mem64	AVX512VL
VBROADCASTSD	zmmreg|mask|z,mem64	AVX512
VBROADCASTSD	Zmmreg | mask | z， mem64	AVX512
VBROADCASTSD	ymmreg|mask|z,xmmreg	AVX512VL
VBROADCASTSD	Ymmreg | mask | z， xmreg	AVX512VL
VBROADCASTSD	zmmreg|mask|z,xmmreg	AVX512
VBROADCASTSD	Zmmreg | mask | z， xmmreg	AVX512
VBROADCASTSS Vbroadcasts.vbroadc ast	xmmreg|mask|z,mem32 Xmmreg | mask | z， mem32	AVX512VL AVX512VL
VBROADCASTSS Vbroadcasts.vbroadc ast	ymmreg|mask|z,mem32 Ymmreg | mask | z， mem32	AVX512VL AVX512VL
VBROADCASTSS Vbroadcasts.vbroadc ast	zmmreg|mask|z,mem32 Zmmreg | mask | z， mem32	AVX512 AVX512
VBROADCASTSS Vbroadcasts.vbroadc ast	xmmreg|mask|z,xmmreg | mask | z， xmmreg	AVX512VL AVX512VL

VBROADCASTSS		
Vbroadcasts.vbroa	ymmreg|mask|z,xmmreg	AVX512VL
dcast	Ymmreg | mask | z，xmreg	AVX512VL
VBROADCASTSS		
Vbroadcasts.vbroa	zmmreg|mask|z,xmmreg	AVX512
dcast	Zmmreg | mask | z，xmmreg	AVX512
	kreg|mask,xmmreg,xmmrm128|b64,imm8 AVX512VL	
VCMPPD	Kreg | mask，xmmreg，xmmrm128 | b64，imm8	
VCMPPD	AVX512VL	
	kreg|mask,ymmreg,ymmrm256|b64,imm8 AVX512VL	
VCMPPD	Kreg | mask，ymmreg，ymmrm256 | b64，imm8	
VCMPPD	AVX512VL	
	kreg|mask,zmmreg,zmmrm512|b64|sae,imm8 AVX512	
VCMPPD	克雷格 | 面具，zmmreg，zmmrm512 | b64 | sae，imm8	
VCMPPD	AVX512	
	kreg|mask,xmmreg,xmmrm128|b32,imm8 AVX512VL	
VCMPPS	Kreg | mask，xmmreg，xmmrm128 | b32，imm8	
VCMPPS	AVX512VL	
	kreg|mask,ymmreg,ymmrm256|b32,imm8 AVX512VL	
VCMPPS	Kreg | mask，ymmreg，ymmrm256 | b32，imm8	
VCMPPS	AVX512VL	
	kreg|mask,zmmreg,zmmrm512|b32|sae,imm8 AVX512	
VCMPPS	克雷格 | 面具，zmmreg，zmmrm512 | b32 | sae，imm8	
VCMPPS	AVX512	
VCMPSD	kreg|mask,xmmreg,xmmrm64|sae,imm8 AVX512	
VCMPSD	Kreg | mask，xmmreg，xmmrm64 | sae，imm8 AVX512	
VCMPSS	kreg|mask,xmmreg,xmmrm32|sae,imm8 AVX512	
VCMPSS	克雷格 | 面具，xmmreg，xmmrm32 | sae，imm8 AVX512	
VCOMISD	xmmreg,xmmrm64|sae	AVX512
VCOMISD	Xmmrm64 | sae	AVX512
VCOMISS	xmmreg,xmmrm32|sae	AVX512
VCOMISS	Xmmreg，xmmrm32 | sae	AVX512
VCOMPRESSPD	mem128|mask,xmmreg	AVX512VL
VCOMPRESSPD	Mem128 | mask，xmmreg	AVX512VL
VCOMPRESSPD	mem256|mask,ymmreg	AVX512VL
VCOMPRESSPD	Mem256 | 面具，ymmreg	AVX512VL
VCOMPRESSPD	mem512|mask,zmmreg	AVX512
VCOMPRESSPD	Mem512 | mask，zmmreg	AVX512
VCOMPRESSPD	xmmreg|mask|z,xmmreg	AVX512VL
VCOMPRESSPD	| mask | z，xmmreg	AVX512VL
VCOMPRESSPD	ymmreg|mask|z,ymmreg	AVX512VL
VCOMPRESSPD	Ymmreg | mask | z，ymmreg	AVX512VL
VCOMPRESSPD	zmmreg|mask|z,zmmreg	AVX512
VCOMPRESSPD	Zmmreg | mask | z，zmmreg	AVX512
VCOMPRESSPS		
VCOMPRESSPS 压	mem128|mask,xmmreg	AVX512VL
缩机	Mem128 | mask，xmmreg	AVX512VL
VCOMPRESSPS		
VCOMPRESSPS 压	mem256|mask,ymmreg	AVX512VL
缩机	Mem256 | 面具，ymmreg	AVX512VL

VCOMPRESSPS
VCOMPRESSPS 压　mem512|mask,zmmreg　AVX512
缩机　Mem512 | mask，zmmreg　AVX512
VCOMPRESSPS
VCOMPRESSPS 压　xmmreg|mask|z,xmmreg　AVX512VL
缩机　| mask | z，xmmreg　AVX512VL
VCOMPRESSPS
VCOMPRESSPS 压　ymmreg|mask|z,ymmreg　AVX512VL
缩机　Ymmreg | mask | z，ymmreg　AVX512VL
VCOMPRESSPS
VCOMPRESSPS 压　zmmreg|mask|z,zmmreg　AVX512
缩机　Zmmreg | mask | z，zmmreg　AVX512
VCVTDQ2PD　xmmreg|mask|z,xmmrm64|b32 AVX512VL
VCVTDQ2PD　Xmmrreg | mask | z，xmmrm64 | b32 AVX512VL
VCVTDQ2PD　ymmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTDQ2PD　Ymmreg | mask | z，xmmrm128 | b32 AVX512VL
VCVTDQ2PD　zmmreg|mask|z,ymmrm256|b32|er AVX512
VCVTDQ2PD　Zmmreg | mask | z，ymmrm256 | b32 | er AVX512
VCVTDQ2PS　xmmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTDQ2PS　Xmmreg | mask | z，xmmrm128 | b32 AVX512VL
VCVTDQ2PS　ymmreg|mask|z,ymmrm256|b32 AVX512VL
VCVTDQ2PS　Ymmreg | mask | z，ymmrm256 | b32 AVX512VL
VCVTDQ2PS　zmmreg|mask|z,zmmrm512|b32|er AVX512
VCVTDQ2PS　Zmmrreg | mask | z，zmmrm512 | b32 | er AVX512
VCVTPD2DQ　xmmreg|mask|z,xmmrm128|b64 AVX512VL
VCVTPD2DQ　Xmmreg | mask | z，xmmrm128 | b64 AVX512VL
VCVTPD2DQ　xmmreg|mask|z,ymmrm256|b64 AVX512VL
VCVTPD2DQ　Xmmreg | mask | z，ymmrm256 | b64 AVX512VL
VCVTPD2DQ　ymmreg|mask|z,zmmrm512|b64|er AVX512
VCVTPD2DQ　Ymmreg | mask | z，zmmrm512 | b64 | er AVX512
VCVTPD2PS　xmmreg|mask|z,xmmrm128|b64 AVX512VL
VCVTPD2PS　Xmmreg | mask | z，xmmrm128 | b64 AVX512VL
VCVTPD2PS　xmmreg|mask|z,ymmrm256|b64 AVX512VL
VCVTPD2PS　Xmmreg | mask | z，ymmrm256 | b64 AVX512VL
VCVTPD2PS　ymmreg|mask|z,zmmrm512|b64|er AVX512
VCVTPD2PS　Ymmreg | mask | z，zmmrm512 | b64 | er AVX512
VCVTPD2QQ　xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ
VCVTPD2QQ　Xmmreg | mask | z，xmmrm128 | b64 AVX512VL/DQ
VCVTPD2QQ　ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ
VCVTPD2QQ　Ymmreg | mask | z，ymmrm256 | b64 AVX512VL/DQ
VCVTPD2QQ　zmmreg|mask|z,zmmrm512|b64|er AVX512DQ
VCVTPD2QQ　Zmmrreg | mask | z，zmmrm512 | b64 | er AVX512DQ
VCVTPD2UDQ　xmmreg|mask|z,xmmrm128|b64 AVX512VL
VCVTPD2UDQ　Xmmreg | mask | z，xmmrm128 | b64 AVX512VL
VCVTPD2UDQ　xmmreg|mask|z,ymmrm256|b64 AVX512VL
VCVTPD2UDQ　Xmmreg | mask | z，ymmrm256 | b64 AVX512VL
VCVTPD2UDQ　ymmreg|mask|z,zmmrm512|b64|er AVX512
VCVTPD2UDQ　Ymmreg | mask | z，zmmrm512 | b64 | er AVX512
VCVTPD2UQQ　xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ
VCVTPD2UQQ　Xmmreg | mask | z，xmmrm128 | b64 AVX512VL/DQ

VCVTPD2UQQ	ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ	
VCVTPD2UQQ	Ymmreg | mask | z，ymmrm256 | b64 AVX512VL/DQ	
VCVTPD2UQQ	zmmreg|mask|z,zmmrm512|b64|er AVX512DQ	
VCVTPD2UQQ	Zmmrreg | mask | z，zmmrm512 | b64 | er AVX512DQ	
VCVTPH2PS	xmmreg|mask|z,xmmrm64	
VCVTPH2PS	Xmmreg | mask | z，	AVX512VL
VCVTPH2PS	xmmrm64	AVX512VL
VCVTPH2PS	ymmreg|mask|z,xmmrm128	
VCVTPH2PS	Ymmreg | mask | z，	AVX512VL
VCVTPH2PS	xmmrm128	AVX512VL
VCVTPH2PS		
VCVTPH2PS	zmmreg|mask|z,ymmrm256|sae AVX512	
VCVTPH2PS	Zmmreg | mask | z，ymmrm256 | sae AVX512	
VCVTPS2DQ	xmmreg|mask|z,xmmrm128|b32 AVX512VL	
VCVTPS2DQ	Xmmreg | mask | z，xmmrm128 | b32 AVX512VL	
VCVTPS2DQ	ymmreg|mask|z,ymmrm256|b32 AVX512VL	
VCVTPS2DQ	Ymmreg | mask | z，ymmrm256 | b32 AVX512VL	
VCVTPS2DQ	zmmreg|mask|z,zmmrm512|b32|er AVX512	
VCVTPS2DQ	Zmmrreg | mask | z，zmmrm512 | b32 | er AVX512	
VCVTPS2PD	xmmreg|mask|z,xmmrm64|b32 AVX512VL	
VCVTPS2PD	Xmmrreg | mask | z，xmmrm64 | b32 AVX512VL	
VCVTPS2PD	ymmreg|mask|z,xmmrm128|b32 AVX512VL	
VCVTPS2PD	Ymmreg | mask | z，xmmrm128 | b32 AVX512VL	
VCVTPS2PD	zmmreg|mask|z,ymmrm256|b32|sae AVX512	
VCVTPS2PD	Zmmreg | mask | z，ymmrm256 | b32 | sae AVX512	

203

VCVTPS2PH	xmmreg|mask|z,xmmreg,imm8 AVX512VL	
VCVTPS2PH	Xmmreg | mask | z， xmmreg， imm8 AVX512VL	
VCVTPS2PH	xmmreg|mask|z,ymmreg,imm8 AVX512VL	
VCVTPS2PH	Xmreg | mask | z， ymmreg， imm8 AVX512VL	
VCVTPS2PH	ymmreg|mask|z,zmmreg|sae,imm8 AVX512	
VCVTPS2PH	Ymmreg | mask | z， zmreg | sae， imm8 AVX512	
	mem64|mask,xmmreg,imm8	
VCVTPS2PH	Mem64 | mask， xmmreg，	AVX512VL
VCVTPS2PH	imm8	AVX512VL
	mem128|mask,ymmreg,imm8	
VCVTPS2PH	Mem128 | mask， ymmreg，	AVX512VL
VCVTPS2PH	imm8	AVX512VL
VCVTPS2PH	mem256|mask,zmmreg|sae,imm8 AVX512	
VCVTPS2PH	Mem256 | mask， zmmreg | sae， imm8 AVX512	
	xmmreg|mask|z,xmmrm64|b32 AVX512VL/DQ	
VCVTPS2QQ	Xmmreg | mask | z， xmmrm64 | b32	
VCVTPS2QQ	AVX512VL/DQ	
	ymmreg|mask|z,xmmrm128|b32 AVX512VL/DQ	
VCVTPS2QQ	Ymmreg | mask | z， xmmrm128 | b32	
VCVTPS2QQ	AVX512VL/DQ	
	zmmreg|mask|z,ymmrm256|b32|er AVX512DQ	
VCVTPS2QQ	Zmmreg | mask | z， ymmrm256 | b32 | er	
VCVTPS2QQ	AVX512DQ	
VCVTPS2UDQ	xmmreg|mask|z,xmmrm128|b32 AVX512VL	
VCVTPS2UDQ	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL	
VCVTPS2UDQ	ymmreg|mask|z,ymmrm256|b32 AVX512VL	
VCVTPS2UDQ	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL	
	zmmreg|mask|z,zmmrm512|b32|er AVX512	
VCVTPS2UDQ	Zmmrreg | mask | z， zmmrm512 | b32 | er	
VCVTPS2UDQ	AVX512	
	xmmreg|mask|z,xmmrm64|b32 AVX512VL/DQ	
VCVTPS2UQQ	Xmmreg | mask | z， xmmrm64 | b32	
VCVTPS2UQQ	AVX512VL/DQ	
	ymmreg|mask|z,xmmrm128|b32 AVX512VL/DQ	
VCVTPS2UQQ	Ymmreg | mask | z， xmmrm128 | b32	
VCVTPS2UQQ	AVX512VL/DQ	
	zmmreg|mask|z,ymmrm256|b32|er AVX512DQ	
VCVTPS2UQQ	Zmmreg | mask | z， ymmrm256 | b32 | er	
VCVTPS2UQQ	AVX512DQ	
	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ	
VCVTQQ2PD	Xmmreg | mask | z， xmmrm128 | b64	
VCVTQQ2PD	AVX512VL/DQ	
	ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ	
VCVTQQ2PD	Ymmreg | mask | z， ymmrm256 | b64	
VCVTQQ2PD	AVX512VL/DQ	
	zmmreg|mask|z,zmmrm512|b64|er AVX512DQ	
VCVTQQ2PD	Zmmrreg | mask | z， zmmrm512 | b64 | er	
VCVTQQ2PD	AVX512DQ	
VCVTQQ2PS	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ	
Vcvtq2ps	Xmmreg | mask | z， xmmrm128 | b64	

	AVX512VL/DQ	
	xmmreg|mask|z,ymmrm256|b64 AVX512VL/DQ	
VCVTQQ2PS	Xmmreg | mask | z， ymmrm256 | b64	
Vcvtq2ps	AVX512VL/DQ	
	ymmreg|mask|z,zmmrm512|b64|er AVX512DQ	
VCVTQQ2PS	Ymmreg | mask | z， zmmrm512 | b64 | er	
Vcvtq2ps	AVX512DQ	
VCVTSD2SI	reg32,xmmrm64|er	AVX512
VCVTSD2SI	Reg32， xmmrm64 | er	AVX512
VCVTSD2SI	reg64,xmmrm64|er	AVX512
VCVTSD2SI	Reg64， xmmrm64 | er	AVX512
	xmmreg|mask|z,xmmreg,xmmrm64|er AVX512	
VCVTSD2SS	Xmmreg | mask | z， xmmreg， xmmrm64 | er	
VCVTSD2SS	AVX512	
VCVTSD2USI	reg32,xmmrm64|er	AVX512
VCVTSD2USI	Reg32， xmmrm64 | er	AVX512
VCVTSD2USI	reg64,xmmrm64|er	AVX512
VCVTSD2USI	Reg64， xmmrm64 | er	AVX512
VCVTSI2SD	xmmreg,xmmreg|er,rm32	AVX512
VCVTSI2SD	Xmmreg， xmmreg | er， rm32	AVX512
VCVTSI2SD	xmmreg,xmmreg|er,rm64	AVX512
VCVTSI2SD	Xmmreg， xmmreg | er， rm64	AVX512
VCVTSI2SS		
VCVTSI2SS	xmmreg,xmmreg|er,rm32	AVX512
VCVTSI2SS	Xmmreg， xmmreg | er， rm32	AVX512
VCVTSI2SS		
VCVTSI2SS	xmmreg,xmmreg|er,rm64	AVX512
VCVTSI2SS	Xmmreg， xmmreg | er， rm64	AVX512
	xmmreg|mask|z,xmmreg,xmmrm32|sae AVX512	
VCVTSS2SD	Xmmreg | mask | z， xmmreg， xmmrm32 | sae	
VCVTSS2SD	AVX512	
VCVTSS2SI	reg32,xmmrm32|er	AVX512
VCVTSS2SI	Reg32， xmmrm32 | er	AVX512
VCVTSS2SI	reg64,xmmrm32|er	AVX512
VCVTSS2SI	Reg64， xmmrm32 | er	AVX512
VCVTSS2USI		
VCVTSS2USI	reg32,xmmrm32|er	AVX512
VCVTSS2USI	Reg32， xmmrm32 | er	AVX512
VCVTSS2USI		
VCVTSS2USI	reg64,xmmrm32|er	AVX512
VCVTSS2USI	Reg64， xmmrm32 | er	AVX512
VCVTTPD2DQ	xmmreg|mask|z,xmmrm128|b64 AVX512VL	
VCVTTPD2DQ	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL	
VCVTTPD2DQ	xmmreg|mask|z,ymmrm256|b64 AVX512VL	
VCVTTPD2DQ	Xmmreg | mask | z， ymmrm256 | b64 AVX512VL	
	ymmreg|mask|z,zmmrm512|b64|sae AVX512	
VCVTTPD2DQ	Ymmreg | mask | z， zmmrm512 | b64 | sae	
VCVTTPD2DQ	AVX512	
	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ	
VCVTTPD2QQ	Xmmreg | mask | z， xmmrm128 | b64	
VCVTTPD2QQ	AVX512VL/DQ	

	ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ
VCVTTPD2QQ	Ymmreg | mask | z，ymmrm256 | b64
VCVTTPD2QQ	AVX512VL/DQ
	zmmreg|mask|z,zmmrm512|b64|sae AVX512DQ
VCVTTPD2QQ	Zmmrreg | mask | z，zmmrm512 | b64 | sae
VCVTTPD2QQ	AVX512DQ
VCVTTPD2UDQ	xmmreg|mask|z,xmmrm128|b64 AVX512VL
VCVTTPD2UDQ	Xmmreg | mask | z，xmmrm128 | b64 AVX512VL
VCVTTPD2UDQ	xmmreg|mask|z,ymmrm256|b64 AVX512VL
VCVTTPD2UDQ	Xmmreg | mask | z，ymmrm256 | b64 AVX512VL
	ymmreg|mask|z,zmmrm512|b64|sae AVX512
VCVTTPD2UDQ	Ymmreg | mask | z，zmmrm512 | b64 | sae
VCVTTPD2UDQ	AVX512
	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ
VCVTTPD2UQQ	Xmmreg | mask | z，xmmrm128 | b64
VCVTTPD2UQQ	AVX512VL/DQ
	ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ
VCVTTPD2UQQ	Ymmreg | mask | z，ymmrm256 | b64
VCVTTPD2UQQ	AVX512VL/DQ
	zmmreg|mask|z,zmmrm512|b64|sae AVX512DQ
VCVTTPD2UQQ	Zmmrreg | mask | z，zmmrm512 | b64 | sae
VCVTTPD2UQQ	AVX512DQ
VCVTTPS2DQ	xmmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTTPS2DQ	Xmmreg | mask | z，xmmrm128 | b32 AVX512VL
VCVTTPS2DQ	ymmreg|mask|z,ymmrm256|b32 AVX512VL
VCVTTPS2DQ	Ymmreg | mask | z，ymmrm256 | b32 AVX512VL
	zmmreg|mask|z,zmmrm512|b32|sae AVX512
VCVTTPS2DQ	Zmmrreg | mask | z，zmmrm512 | b32 | sae
VCVTTPS2DQ	AVX512
	xmmreg|mask|z,xmmrm64|b32 AVX512VL/DQ
VCVTTPS2QQ	Xmmreg | mask | z，xmmrm64 | b32
VCVTTPS2QQ	AVX512VL/DQ
	ymmreg|mask|z,xmmrm128|b32 AVX512VL/DQ
VCVTTPS2QQ	Ymmreg | mask | z，xmmrm128 | b32
VCVTTPS2QQ	AVX512VL/DQ
	zmmreg|mask|z,ymmrm256|b32|sae AVX512DQ
VCVTTPS2QQ	Zmmreg | mask | z，ymmrm256 | b32 | sae
VCVTTPS2QQ	AVX512DQ
VCVTTPS2UDQ	xmmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTTPS2UDQ	Xmmreg | mask | z，xmmrm128 | b32 AVX512VL

VCVTTPS2UDQ	ymmreg|mask|z,ymmrm256|b32 AVX512VL
VCVTTPS2UDQ	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL
VCVTTPS2UDQ	zmmreg|mask|z,zmmrm512|b32|sae AVX512
VCVTTPS2UDQ	Zmmrreg | mask | z， zmmrm512 | b32 | sae AVX512
VCVTTPS2UQQ	xmmreg|mask|z,xmmrm64|b32 AVX512VL/DQ
VCVTTPS2UQQ	Xmmreg | mask | z， xmmrm64 | b32 AVX512VL/DQ
VCVTTPS2UQQ	ymmreg|mask|z,xmmrm128|b32 AVX512VL/DQ
VCVTTPS2UQQ	Ymmreg | mask | z， xmmrm128 | b32 AVX512VL/DQ
VCVTTPS2UQQ	zmmreg|mask|z,ymmrm256|b32|sae AVX512DQ
VCVTTPS2UQQ	Zmmreg | mask | z， ymmrm256 | b32 | sae AVX512DQ
VCVTTSD2SI	reg32,xmmrm64|sae AVX512
VCVTTSD2SI	Reg32， xmmrm64 | sae AVX512
VCVTTSD2SI	reg64,xmmrm64|sae AVX512
VCVTTSD2SI	Reg64， xmmrm64 | sae AVX512
VCVTTSD2USI	reg32,xmmrm64|sae AVX512
VCVTTSD2USI	Reg32， xmmrm64 | sae AVX512
VCVTTSD2USI	reg64,xmmrm64|sae AVX512
VCVTTSD2USI	Reg64， xmmrm64 | sae AVX512
VCVTTSS2SI	reg32,xmmrm32|sae AVX512
VCVTTSS2SI	Reg32， xmmrm32 | sae AVX512
VCVTTSS2SI	reg64,xmmrm32|sae AVX512
VCVTTSS2SI	Reg64， xmmrm32 | sae AVX512
VCVTTSS2USI	reg32,xmmrm32|sae AVX512
VCVTTSS2USI	Reg32， xmmrm32 | sae AVX512
VCVTTSS2USI	reg64,xmmrm32|sae AVX512
VCVTTSS2USI	Reg64， xmmrm32 | sae AVX512
VCVTUDQ2PD	xmmreg|mask|z,xmmrm64|b32 AVX512VL
VCVTUDQ2PD	Xmmrreg | mask | z， xmmrm64 | b32 AVX512VL
VCVTUDQ2PD	ymmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTUDQ2PD	Ymmreg | mask | z， xmmrm128 | b32 AVX512VL
VCVTUDQ2PD	zmmreg|mask|z,ymmrm256|b32|er AVX512
VCVTUDQ2PD	Zmmreg | mask | z， ymmrm256 | b32 | er AVX512
VCVTUDQ2PS	xmmreg|mask|z,xmmrm128|b32 AVX512VL
VCVTUDQ2PS	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL
VCVTUDQ2PS	ymmreg|mask|z,ymmrm256|b32 AVX512VL
VCVTUDQ2PS	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL
VCVTUDQ2PS	zmmreg|mask|z,zmmrm512|b32|er AVX512
VCVTUDQ2PS	Zmmrreg | mask | z， zmmrm512 | b32 | er AVX512
VCVTUQQ2PD	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ
VCVTUQQ2PD	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL/DQ
VCVTUQQ2PD	ymmreg|mask|z,ymmrm256|b64 AVX512VL/DQ
VCVTUQQ2PD	Ymmreg | mask | z， ymmrm256 | b64 AVX512VL/DQ
VCVTUQQ2PD	zmmreg|mask|z,zmmrm512|b64|er AVX512DQ
VCVTUQQ2PD	Zmmrreg | mask | z， zmmrm512 | b64 | er AVX512DQ
VCVTUQQ2PS	xmmreg|mask|z,xmmrm128|b64 AVX512VL/DQ
VCVTUQQ2PS	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL/DQ
VCVTUQQ2PS	xmmreg|mask|z,ymmrm256|b64 AVX512VL/DQ
VCVTUQQ2PS	Xmmreg | mask | z， ymmrm256 | b64 AVX512VL/DQ
VCVTUQQ2PS	ymmreg|mask|z,zmmrm512|b64|er AVX512DQ
VCVTUQQ2PS	Ymmreg | mask | z， zmmrm512 | b64 | er AVX512DQ

	xmmreg,xmmreg|er,rm32	
VCVTUSI2SD	Xmmreg，xmmreg | er，	AVX512
VCVTUSI2SD	rm32	AVX512
	xmmreg,xmmreg|er,rm64	
VCVTUSI2SD	Xmmreg，xmmreg | er，	AVX512
VCVTUSI2SD	rm64	AVX512
	xmmreg,xmmreg|er,rm32	
VCVTUSI2SS	Xmmreg，xmmreg | er，	AVX512
VCVTUSI2SS	rm32	AVX512
	xmmreg,xmmreg|er,rm64	
VCVTUSI2SS	Xmmreg，xmmreg | er，	AVX512
VCVTUSI2SS	rm64	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128,imm8 AVX512VL/BW	
VDBPSADBW	Xmmreg | mask | z，xmmreg *，xmmrm128，imm8	
VDBPSADBW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256,imm8 AVX512VL/BW	
VDBPSADBW	Ymmreg | mask | z，ymmreg *，ymmrm256，imm8	
VDBPSADBW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512,imm8 AVX512BW	
VDBPSADBW	Zmmreg | mask | z，zmmreg *，zmmrm512，imm8	
VDBPSADBW	AVX512BW	
VDIVPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VDIVPD	Xmmreg | mask | z，xmmreg *，xmmrm128 | b64 AVX512VL	
VDIVPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VDIVPD	Ymmreg | mask | z，ymmreg *，ymmrm256 | b64 AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|er AVX512	
VDIVPD	Zmmreg | mask | z，zmmreg *，zmmrm512 | b64 | er	
VDIVPD	AVX512	
VDIVPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL	
VDIVPS	Xmmreg | mask | z，xmmreg *，xmmrm128 | b32 AVX512VL	
VDIVPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL	
VDIVPS	Ymmreg | mask | z，ymmreg *，ymmrm256 | b32 AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|er AVX512	
VDIVPS	Zmmreg | mask | z，zmmreg *，zmmrm512 | b32 | er	
VDIVPS	AVX512	
VDIVSD	xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512	
VDIVSD	Xmmreg | mask | z，xmmreg *，xmmrm64 | er AVX512	
VDIVSS	xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512	
VDIVSS	Xmmreg | mask | z，xmmreg *，xmmrm32 | er AVX512	
VEXP2PD	zmmreg|mask|z,zmmrm512|b64|sae AVX512ER	
VEXP2PD	Zmmrreg | mask | z，zmmrm512 | b64 | sae AVX512ER	
VEXP2PS	zmmreg|mask|z,zmmrm512|b32|sae AVX512ER	
VEXP2PS	Zmmrreg | mask | z，zmmrm512 | b32 | sae AVX512ER	
VEXPANDPD	xmmreg|mask|z,mem128	AVX512VL
VEXPANDPD	Xmmreg | mask | z，mem128	AVX512VL
VEXPANDPD	ymmreg|mask|z,mem256	AVX512VL
VEXPANDPD	Ymmreg | mask | z，mem256	AVX512VL
VEXPANDPD	zmmreg|mask|z,mem512	AVX512
VEXPANDPD	Zmmreg | mask | z，mem512	AVX512
VEXPANDPD		AVX512VL
VEXPANDPD	xmmreg|mask|z,xmmreg	AVX512VL

	| mask | z， xmmreg	
VEXPANDPD	ymmreg|mask|z,ymmreg	AVX512VL
VEXPANDPD	Ymmreg | mask | z， ymmreg	AVX512VL
VEXPANDPD	zmmreg|mask|z,zmmreg	AVX512
VEXPANDPD	Zmmreg | mask | z， zmmreg	AVX512
VEXPANDPS	xmmreg|mask|z,mem128	AVX512VL
VEXPANDPS	Xmmreg | mask | z， mem128	AVX512VL
VEXPANDPS	ymmreg|mask|z,mem256	AVX512VL
VEXPANDPS	Ymmreg | mask | z， mem256	AVX512VL
VEXPANDPS	zmmreg|mask|z,mem512	AVX512
VEXPANDPS	Zmmreg | mask | z， mem512	AVX512
VEXPANDPS	xmmreg|mask|z,xmmreg	AVX512VL
VEXPANDPS	| mask | z， xmmreg	AVX512VL
VEXPANDPS	ymmreg|mask|z,ymmreg	AVX512VL
VEXPANDPS	Ymmreg | mask | z， ymmreg	AVX512VL
VEXPANDPS	zmmreg|mask|z,zmmreg	AVX512
VEXPANDPS	Zmmreg | mask | z， zmmreg	AVX512

205

VEXTRACTF32X4	xmmreg|mask|z,ymmreg,imm8	AVX512VL
VEXTRACTF32X4	Xmreg | mask | z， ymmreg， imm8	AVX512VL
VEXTRACTF32X4	xmmreg|mask|z,zmmreg,imm8	AVX512
VEXTRACTF32X4	Xmmreg | mask | z， zmmreg， imm8	AVX512
	mem128|mask,ymmreg,imm8	
VEXTRACTF32X4	Mem128 | mask， ymmreg，	AVX512VL
VEXTRACTF32X4	imm8	AVX512VL
	mem128|mask,zmmreg,imm8	
VEXTRACTF32X4	Mem128 | mask， zmmreg，	AVX512
VEXTRACTF32X4	imm8	AVX512
VEXTRACTF32X8	ymmreg|mask|z,zmmreg,imm8	AVX512DQ
VEXTRACTF32X8	Ymmreg | mask | z， zmreg， imm8	AVX512DQ
	mem256|mask,zmmreg,imm8	
VEXTRACTF32X8	Mem256 | mask， zmmreg，	AVX512DQ
VEXTRACTF32X8	imm8	AVX512DQ
VEXTRACTF64X2	xmmreg|mask|z,ymmreg,imm8	AVX512VL/DQ
VEXTRACTF64X2	Xmreg | mask | z， ymmreg， imm8	AVX512VL/DQ
VEXTRACTF64X2	xmmreg|mask|z,zmmreg,imm8	AVX512DQ
VEXTRACTF64X2	Xmmreg | mask | z， zmmreg， imm8	AVX512DQ
	mem128|mask,ymmreg,imm8	
VEXTRACTF64X2	Mem128 | mask， ymmreg，	AVX512VL/DQ
VEXTRACTF64X2	imm8	AVX512VL/DQ
	mem128|mask,zmmreg,imm8	
VEXTRACTF64X2	Mem128 | mask， zmmreg，	AVX512DQ
VEXTRACTF64X2	imm8	AVX512DQ
VEXTRACTF64X4	ymmreg|mask|z,zmmreg,imm8	AVX512
VEXTRACTF64X4	Ymmreg | mask | z， zmreg， imm8	AVX512
	mem256|mask,zmmreg,imm8	
VEXTRACTF64X4	Mem256 | mask， zmmreg，	AVX512
VEXTRACTF64X4	imm8	AVX512
VEXTRACTI32X4	xmmreg|mask|z,ymmreg,imm8	AVX512VL
VEXTRACTI32X4	Xmreg | mask | z， ymmreg， imm8	AVX512VL
VEXTRACTI32X4	xmmreg|mask|z,zmmreg,imm8	AVX512
VEXTRACTI32X4	Xmmreg | mask | z， zmmreg， imm8	AVX512
	mem128|mask,ymmreg,imm8	
VEXTRACTI32X4	Mem128 | mask， ymmreg，	AVX512VL
VEXTRACTI32X4	imm8	AVX512VL
	mem128|mask,zmmreg,imm8	
VEXTRACTI32X4	Mem128 | mask， zmmreg，	AVX512
VEXTRACTI32X4	imm8	AVX512
VEXTRACTI32X8	ymmreg|mask|z,zmmreg,imm8	AVX512DQ
VEXTRACTI32X8	Ymmreg | mask | z， zmreg， imm8	AVX512DQ
	mem256|mask,zmmreg,imm8	
VEXTRACTI32X8	Mem256 | mask， zmmreg，	AVX512DQ
VEXTRACTI32X8	imm8	AVX512DQ
VEXTRACTI64X2	xmmreg|mask|z,ymmreg,imm8	AVX512VL/DQ
VEXTRACTI64X2	Xmreg | mask | z， ymmreg， imm8	AVX512VL/DQ
VEXTRACTI64X2	xmmreg|mask|z,zmmreg,imm8	AVX512DQ
VEXTRACTI64X2	Xmmreg | mask | z， zmmreg， imm8	AVX512DQ
VEXTRACTI64X2	mem128|mask,ymmreg,imm8	AVX512VL/DQ

VEXTRACTI64X2	Mem128 | mask，ymmreg，imm8	AVX512VL/DQ
	mem128|mask,zmmreg,imm8	
VEXTRACTI64X2	Mem128 | mask，zmmreg，	AVX512DQ
VEXTRACTI64X2	imm8	AVX512DQ
VEXTRACTI64X4	ymmreg|mask|z,zmmreg,imm8 AVX512	
VEXTRACTI64X4	Ymmreg | mask | z，zmreg，imm8 AVX512	
	mem256|mask,zmmreg,imm8	
VEXTRACTI64X4	Mem256 | mask，zmmreg，	AVX512
VEXTRACTI64X4	imm8	AVX512
VEXTRACTPS	reg32,xmmreg,imm8	AVX512
VEXTRACTPS	Reg32，xmmreg，imm8	AVX512
VEXTRACTPS	reg64,xmmreg,imm8	AVX512
VEXTRACTPS	Reg64，xmmreg，imm8	AVX512
VEXTRACTPS	mem32,xmmreg,imm8	AVX512
VEXTRACTPS	Mem32，xmmreg，imm8	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VL	
VFIXUPIMMPD	Xmmreg | mask | z，xmmreg *，xmmrm128 | b64，imm8	
VFIXUPIMMPD	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL	
VFIXUPIMMPD	Ymmreg | mask | z，ymmreg *，ymmrm256 | b64，imm8	
VFIXUPIMMPD	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|sae,imm8 AVX512	
VFIXUPIMMPD	Zmmreg | mask | z，zmmreg *，zmmrm512 | b64 | sae，imm8	
VFIXUPIMMPD	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8 AVX512VL	
VFIXUPIMMPS	Xmmreg | mask | z，xmmreg *，xmmrm128 | b32，imm8	
VFIXUPIMMPS	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL	
VFIXUPIMMPS	Ymmreg | mask | z，ymmreg *，ymmrm256 | b32，imm8	
VFIXUPIMMPS	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|sae,imm8 AVX512	
VFIXUPIMMPS	Zmmreg | mask | z，zmmreg *，zmmrm512 | b32 | sae，imm8	
VFIXUPIMMPS	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm64|sae,imm8 AVX512	
VFIXUPIMMSD	Xmmreg | mask | z，xmmreg *，xmmrm64 | sae，imm8	
VFIXUPIMMSD	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm32|sae,imm8 AVX512	
VFIXUPIMMSS	Xmmreg | mask | z，xmmreg *，xmmrm32 | sae，imm8	
VFIXUPIMMSS	AVX512	
VFMADD132PD	xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL	
VFMADD132PD	Xmmreg | mask | z，xmmreg，xmmrm128 | b64 AVX512VL	
VFMADD132PD	ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL	
VFMADD132PD	Ymmreg | mask | z，ymmreg，ymmrm256 | b64 AVX512VL	
VFMADD132PD	zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512	
VFMADD132PD	Zmmreg | mask | z，zmmreg，zmmrm512 | b64 | er AVX512	
VFMADD132PS	xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL	
VFMADD132PS	Xmmreg | mask | z，xmmreg，xmmrm128 | b32 AVX512VL	
VFMADD132PS	ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL	
VFMADD132PS	Ymmreg | mask | z，ymmreg，ymmrm256 | b32 AVX512VL	
VFMADD132PS	zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512	

VFMADD132PS	Zmmreg | mask | z，zmmreg，zmmrm512 | b32 | er AVX512
VFMADD132SD	xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMADD132SD	Xmmreg | mask | z，xmmreg，xmmrm64 | er AVX512
VFMADD132SS	xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMADD132SS	Xmmreg | mask | z，xmmreg，xmmrm32 | er AVX512
VFMADD213PD	xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMADD213PD	Xmmreg | mask | z，xmmreg，xmmrm128 | b64 AVX512VL
VFMADD213PD	ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMADD213PD	Ymmreg | mask | z，ymmreg，ymmrm256 | b64 AVX512VL
VFMADD213PD	zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMADD213PD	Zmmreg | mask | z，zmmreg，zmmrm512 | b64 | er AVX512
VFMADD213PS	xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMADD213PS	Xmmreg | mask | z，xmmreg，xmmrm128 | b32 AVX512VL
VFMADD213PS	ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMADD213PS	Ymmreg | mask | z，ymmreg，ymmrm256 | b32 AVX512VL
VFMADD213PS	zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMADD213PS	Zmmreg | mask | z，zmmreg，zmmrm512 | b32 | er AVX512
VFMADD213SD	xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMADD213SD	Xmmreg | mask | z，xmmreg，xmmrm64 | er AVX512
VFMADD213SS	xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMADD213SS	Xmmreg | mask | z，xmmreg，xmmrm32 | er AVX512
VFMADD231PD	xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMADD231PD	Xmmreg | mask | z，xmmreg，xmmrm128 | b64 AVX512VL
VFMADD231PD	ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMADD231PD	Ymmreg | mask | z，ymmreg，ymmrm256 | b64 AVX512VL
VFMADD231PD	zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMADD231PD	Zmmreg | mask | z，zmmreg，zmmrm512 | b64 | er AVX512

206

```
VFMADD231PS      xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMADD231PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFMADD231PS      ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMADD231PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFMADD231PS      zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMADD231PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFMADD231SD      xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMADD231SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFMADD231SS      xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMADD231SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFMADDSUB132PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMADDSUB132PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMADDSUB132PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMADDSUB132PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMADDSUB132PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMADDSUB132PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMADDSUB213PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMADDSUB213PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMADDSUB213PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMADDSUB213PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMADDSUB213PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMADDSUB213PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMADDSUB231PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMADDSUB231PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMADDSUB231PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMADDSUB231PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMADDSUB231PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMADDSUB231PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMADDSUB132PD xmmreg | 面具 | Z, xmmreg, xmmrm128 | B64
AVX512VL VFMADDSUB132PD ymmreg | 面具 | Z, ymmreg, ymmrm256 |
B64 AVX512VL VFMADDSUB132PD zmmreg | 面具 | Z, zmreg,
zmmrm512 | B64 | Er AVX512 VFMADDSUB132PS xmmreg | 面具 |
Z, xmmreg, xmmrm128 | B32 AVX512VL VFMADDSUB132PS ymmreg |
面具 | Z, ymmreg, ymmrm256 | B32 AVX512VL VFMADDSUB132PS
zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512
VFMADDSUB213PD xmmreg | 面具 | Z, xmmreg, xmmrm128 | B64
AVX512VL VFMADDSUB213PD ymmreg | 面具 | Z, ymmreg, ymmrm256 |
B64 AVX512VL VFMADDSUB213PD zmmreg | 面具 | Z, zmreg,
zmmrm512 | B64 | Er AVX512 VFMADDSUB213PS xmmreg | 面具 |
Z, xmmreg, xmmrm128 | B32 AVX512VL VFMADDSUB213PS ymmreg |
面具 | Z, ymmreg, ymmrm256 | B32 AVX512VL VFMADDSUB213PS
zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512
VFMADDSUB231PD xmmreg | 面具 | Z, xmmreg, xmmrm128 | B64
AVX512VL VFMADDSUB231PD ymmreg | 面具 | Z, ymmreg, ymmrm256 |
B64 AVX512VL VFMADDSUB231PD zmmreg | 面具 | Z, zmreg,
zmmrm512 | B64 | Er AVX512 VFMADDSUB231PS xmmreg | 面具 |
Z, xmmreg, xmmrm128 | B32 AVX512VL VFMADDSUB231PS ymmreg |
面具 | Z, ymmreg, ymmrm256 | B32 AVX512VL VFMADDSUB231PS
zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512
VFMSUB132PD      xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUB132PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFMSUB132PD      ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUB132PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFMSUB132PD      zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
```

```
VFMSUB132PD zmmreg | mask | z， zmmreg， zmmrm512 | b64 | er AVX512
VFMSUB132PS      xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUB132PS xmmreg | mask | z， xmmreg， xmmrm128 | b32 AVX512VL
VFMSUB132PS      ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUB132PS ymmreg | mask | z， ymmreg， ymmrm256 | b32 AVX512VL
VFMSUB132PS      zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUB132PS zmmreg | mask | z， zmmreg， zmmrm512 | b32 | er AVX512
VFMSUB132SD      xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMSUB132SD xmmreg | mask | z， xmmreg， xmmrm64 | er AVX512
VFMSUB132SS      xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMSUB132SS xmmreg | mask | z， xmmreg， xmmrm32 | er AVX512
VFMSUB213PD      xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUB213PD xmmreg | mask | z， xmmreg， xmmrm128 | b64 AVX512VL
VFMSUB213PD      ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUB213PD ymmreg | mask | z， ymmreg， ymmrm256 | b64 AVX512VL
VFMSUB213PD      zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMSUB213PD zmmreg | mask | z， zmmreg， zmmrm512 | b64 | er AVX512
VFMSUB213PS      xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUB213PS xmmreg | mask | z， xmmreg， xmmrm128 | b32 AVX512VL
VFMSUB213PS      ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUB213PS ymmreg | mask | z， ymmreg， ymmrm256 | b32 AVX512VL
VFMSUB213PS      zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUB213PS zmmreg | mask | z， zmmreg， zmmrm512 | b32 | er AVX512
VFMSUB213SD      xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMSUB213SD xmmreg | mask | z， xmmreg， xmmrm64 | er AVX512
VFMSUB213SS      xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMSUB213SS xmmreg | mask | z， xmmreg， xmmrm32 | er AVX512
VFMSUB231PD      xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUB231PD xmmreg | mask | z， xmmreg， xmmrm128 | b64 AVX512VL
VFMSUB231PD      ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUB231PD ymmreg | mask | z， ymmreg， ymmrm256 | b64 AVX512VL
VFMSUB231PD      zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMSUB231PD zmmreg | mask | z， zmmreg， zmmrm512 | b64 | er AVX512
VFMSUB231PS      xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUB231PS xmmreg | mask | z， xmmreg， xmmrm128 | b32 AVX512VL
VFMSUB231PS      ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUB231PS ymmreg | mask | z， ymmreg， ymmrm256 | b32 AVX512VL
VFMSUB231PS      zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUB231PS zmmreg | mask | z， zmmreg， zmmrm512 | b32 | er AVX512
VFMSUB231SD      xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFMSUB231SD xmmreg | mask | z， xmmreg， xmmrm64 | er AVX512
VFMSUB231SS      xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFMSUB231SS xmmreg | mask | z， xmmreg， xmmrm32 | er AVX512
VFMSUBADD132PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUBADD132PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUBADD132PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMSUBADD132PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUBADD132PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUBADD132PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUBADD213PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUBADD132PD xmmreg | 面具 | Z， xmmreg， xmmrm128 | B64
AVX512VL VFMSUBADD132PD ymmreg | 面具 | Z， ymmreg， ymmrm256 |
B64 AVX512VL VFMSUBADD132PD zmmreg | 面具 | Z， zmreg，
zmmrm512 | B64 | Er AVX512 VFMSUBADD132PS xmmreg | 面具 |
Z， xmmreg， xmmrm128 | B32 AVX512VL VFMSUBADD132PS ymmreg |
面具 | Z， ymmreg， ymmrm256 | B32 AVX512VL VFMSUBADD132PS
```

zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512
VFMSUBADD213PD xmmreg | 面具 | Z, xmmreg, xmmrm128 | B64
AVX512VL

207

VFMSUBADD213PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUBADD213PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMSUBADD213PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUBADD213PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUBADD213PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUBADD231PD xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFMSUBADD231PD ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFMSUBADD231PD zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFMSUBADD231PS xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFMSUBADD231PS ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFMSUBADD231PS zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFMSUBADD213PD ymmreg | 面具 | Z, ymmreg, ymmrm256 | B64 AVX512VL VFMSUBADD213PD zmmreg | 面具 | Z, zmreg, zmmrm512 | B64 | Er AVX512 VFMSUBADD213PS xmmreg | 面具 | Z, xmmreg, xmmrm128 | B32 AVX512VL VFMSUBADD213PS ymmreg | 面具 | Z, ymmreg, ymmrm256 | B32 AVX512VL VFMSUBADD213PS zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512 VFMSUBADD231PD xmmreg | 面具 | Z, xmmreg, xmmrm128 | B64 AVX512VL VFMSUBADD231PD ymmreg | 面具 | Z, ymmreg, ymmrm256 | B64 AVX512VL VFMSUBADD231PD zmmreg | 面具 | Z, zmreg, zmmrm512 | B64 | Er AVX512 VFMSUBADD231PS xmmreg | 面具 | Z, xmmreg, xmmrm128 | B32 AVX512VL VFMSUBADD231PS ymmreg | 面具 | Z, ymmreg, ymmrm256 | B32 AVX512VL VFMSUBADD231PS zmmreg | 面具 | Z, zmreg, zmmrm512 | B32 | Er AVX512
VFNMADD132PD   xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMADD132PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMADD132PD   ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMADD132PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMADD132PD   zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMADD132PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
VFNMADD132PS   xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFNMADD132PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFNMADD132PS   ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFNMADD132PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFNMADD132PS   zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFNMADD132PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFNMADD132SD   xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFNMADD132SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFNMADD132SS   xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFNMADD132SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFNMADD213PD   xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMADD213PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMADD213PD   ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMADD213PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMADD213PD   zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMADD213PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
VFNMADD213PS   xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFNMADD213PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFNMADD213PS   ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFNMADD213PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFNMADD213PS   zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFNMADD213PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFNMADD213SD   xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFNMADD213SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFNMADD213SS   xmmreg|mask|z,xmmreg,xmmrm32|er AVX512

```
VFNMADD213SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFNMADD231PD     xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMADD231PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMADD231PD     ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMADD231PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMADD231PD     zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMADD231PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
VFNMADD231PS     xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFNMADD231PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFNMADD231PS     ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFNMADD231PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFNMADD231PS     zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFNMADD231PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFNMADD231SD     xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFNMADD231SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFNMADD231SS     xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFNMADD231SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFNMSUB132PD     xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMSUB132PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMSUB132PD     ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMSUB132PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMSUB132PD     zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMSUB132PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
VFNMSUB132PS     xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFNMSUB132PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFNMSUB132PS     ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFNMSUB132PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFNMSUB132PS     zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFNMSUB132PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFNMSUB132SD     xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFNMSUB132SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFNMSUB132SS     xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFNMSUB132SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFNMSUB213PD     xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMSUB213PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMSUB213PD     ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMSUB213PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMSUB213PD     zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMSUB213PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
VFNMSUB213PS     xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL
VFNMSUB213PS xmmreg | mask | z, xmmreg, xmmrm128 | b32 AVX512VL
VFNMSUB213PS     ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL
VFNMSUB213PS ymmreg | mask | z, ymmreg, ymmrm256 | b32 AVX512VL
VFNMSUB213PS     zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512
VFNMSUB213PS zmmreg | mask | z, zmmreg, zmmrm512 | b32 | er AVX512
VFNMSUB213SD     xmmreg|mask|z,xmmreg,xmmrm64|er AVX512
VFNMSUB213SD xmmreg | mask | z, xmmreg, xmmrm64 | er AVX512
VFNMSUB213SS     xmmreg|mask|z,xmmreg,xmmrm32|er AVX512
VFNMSUB213SS xmmreg | mask | z, xmmreg, xmmrm32 | er AVX512
VFNMSUB231PD     xmmreg|mask|z,xmmreg,xmmrm128|b64 AVX512VL
VFNMSUB231PD xmmreg | mask | z, xmmreg, xmmrm128 | b64 AVX512VL
VFNMSUB231PD     ymmreg|mask|z,ymmreg,ymmrm256|b64 AVX512VL
VFNMSUB231PD ymmreg | mask | z, ymmreg, ymmrm256 | b64 AVX512VL
VFNMSUB231PD     zmmreg|mask|z,zmmreg,zmmrm512|b64|er AVX512
VFNMSUB231PD zmmreg | mask | z, zmmreg, zmmrm512 | b64 | er AVX512
```

208

VFNMSUB231PS	xmmreg|mask|z,xmmreg,xmmrm128|b32 AVX512VL	
VFNMSUB231PS	Xmmreg | mask | z， xmmreg， xmmrm128 | b32 AVX512VL	
VFNMSUB231PS	ymmreg|mask|z,ymmreg,ymmrm256|b32 AVX512VL	
VFNMSUB231PS	Ymmreg | mask | z， ymmreg， ymmrm256 | b32 AVX512VL	
VFNMSUB231PS	zmmreg|mask|z,zmmreg,zmmrm512|b32|er AVX512	
VFNMSUB231PS	Zmmreg | mask | z， zmmreg， zmmrm512 | b32 | er AVX512	
VFNMSUB231SD	xmmreg|mask|z,xmmreg,xmmrm64|er AVX512	
VFNMSUB231SD	Xmmreg | mask | z， xmmreg， xmmrm64 | er AVX512	
VFNMSUB231SS	xmmreg|mask|z,xmmreg,xmmrm32|er AVX512	
VFNMSUB231SS	Xmmreg | mask | z， xmmreg， xmmrm32 | er AVX512	
VFPCLASSPD	kreg|mask,xmmrm128|b64,imm8 AVX512VL/DQ	
VFPCLASSPD	Kreg | mask， xmmrm128 | b64， imm8 AVX512VL/DQ	
VFPCLASSPD	kreg|mask,ymmrm256|b64,imm8 AVX512VL/DQ	
VFPCLASSPD	克雷格 | 面具， ymmrm256 | b64， imm8 AVX512VL/DQ	
VFPCLASSPD	kreg|mask,zmmrm512|b64,imm8 AVX512DQ	
VFPCLASSPD	Kreg | mask， zmmrm512 | b64， imm8 AVX512DQ	
VFPCLASSPS	kreg|mask,xmmrm128|b32,imm8 AVX512VL/DQ	
VFPCLASSPS	Kreg | mask， xmmrm128 | b32， imm8 AVX512VL/DQ	
VFPCLASSPS	kreg|mask,ymmrm256|b32,imm8 AVX512VL/DQ	
VFPCLASSPS	克雷格 | 面具， ymmrm256 | b32， imm8 AVX512VL/DQ	
VFPCLASSPS	kreg|mask,zmmrm512|b32,imm8 AVX512DQ	
VFPCLASSPS	Kreg | mask， zmmrm512 | b32， imm8 AVX512DQ	
	kreg|mask,xmmrm64,imm8	
VFPCLASSSD	克雷格 | 面具， xmmrm64，	AVX512DQ
VFPCLASSSD	imm8	AVX512DQ
	kreg|mask,xmmrm32,imm8	
VFPCLASSSS	Kreg | mask， xmmrm32，	AVX512DQ
VFPCLASSSS	imm8	AVX512DQ
VGATHERDPD	xmmreg|mask,xmem64	AVX512VL
Vgather dpd	Xmmreg | mask， xmem64	AVX512VL
VGATHERDPD	ymmreg|mask,xmem64	AVX512VL
Vgather dpd	Ymmreg | mask， xmem64	AVX512VL
VGATHERDPD	zmmreg|mask,ymem64	AVX512
Vgather dpd	Zmmreg | mask， ymem64	AVX512
VGATHERDPS	xmmreg|mask,xmem32	AVX512VL
Vgather dps	Xmmreg | mask， xmem32	AVX512VL
VGATHERDPS	ymmreg|mask,ymem32	AVX512VL
Vgather dps	Ymmreg | mask， ymem32	AVX512VL
VGATHERDPS	zmmreg|mask,zmem32	AVX512
Vgather dps	Zmmreg | mask， zmem32	AVX512
VGATHERPF0DPD	ymem64|mask	AVX512PF
Vgather pf0dpd	Ymem64 | 面具	AVX512PF AVX512PF
VGATHERPF0DPS	zmem32|mask	AVX512PF
好的，谢谢	Zmem32 | 面具	AVX512PF AVX512PF
VGATHERPF0QPD	zmem64|mask	AVX512PF
Vgather pf0qpd	Zmem64 | 面具	AVX512PF AVX512PF
VGATHERPF0QPS	zmem32|mask	AVX512PF
快点，快点	Zmem32 | 面具	AVX512PF AVX512PF
VGATHERPF1DPD	ymem64|mask	AVX512PF
Vgather pf1dpd	Ymem64 | 面具	AVX512PF AVX512PF

VGATHERPF1DPS	zmem32|mask	AVX512PF
Vgather pf1dps	Zmem32 | 面具	AVX512PF AVX512PF
VGATHERPF1QPD	zmem64|mask	AVX512PF
Vgather pf1qpd	Zmem64 | 面具	AVX512PF AVX512PF
VGATHERPF1QPS	zmem32|mask	AVX512PF
Vgather pf1qps	Zmem32 | 面具	AVX512PF AVX512PF
VGATHERQPD	xmmreg|mask,xmem64	AVX512VL
Vgather qpd	Xmmreg | mask， xmem64	AVX512VL
VGATHERQPD	ymmreg|mask,ymem64	AVX512VL
Vgather qpd	Ymmreg | mask， ymem64	AVX512VL
VGATHERQPD	zmmreg|mask,zmem64	AVX512
Vgather qpd	Zmemreg | mask， zmem64	AVX512
VGATHERQPS	xmmreg|mask,xmem32	AVX512VL
Vgather qps	Xmmreg | mask， xmem32	AVX512VL
VGATHERQPS	xmmreg|mask,ymem32	AVX512VL
Vgather qps	Xmmreg | mask， ymem32	AVX512VL
VGATHERQPS	ymmreg|mask,zmem32	AVX512
Vgather qps	Ymmreg | mask， zmem32	AVX512
VGETEXPPD	xmmreg|mask|z,xmmrm128|b64 AVX512VL	
VGETEXPPD	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL	
VGETEXPPD	ymmreg|mask|z,ymmrm256|b64 AVX512VL	
VGETEXPPD	Ymmreg | mask | z， ymmrm256 | b64 AVX512VL	
VGETEXPPD	zmmreg|mask|z,zmmrm512|b64|sae AVX512	
VGETEXPPD	Zmmrreg | mask | z， zmmrm512 | b64 | sae AVX512	
VGETEXPPS	xmmreg|mask|z,xmmrm128|b32 AVX512VL	
VGETEXPPS	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL	
VGETEXPPS	ymmreg|mask|z,ymmrm256|b32 AVX512VL	
VGETEXPPS	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL	
VGETEXPPS	zmmreg|mask|z,zmmrm512|b32|sae AVX512	
VGETEXPPS	Zmmrreg | mask | z， zmmrm512 | b32 | sae AVX512	
VGETEXPSD	xmmreg|mask|z,xmmreg,xmmrm64|sae AVX512	
VGETEXPSD	Xmmreg | mask | z， xmmreg， xmmrm64 | sae AVX512	
VGETEXPSS	xmmreg|mask|z,xmmreg,xmmrm32|sae AVX512	
VGETEXPSS	Xmmreg | mask | z， xmmreg， xmmrm32 | sae AVX512	
VGETMANTPD	xmmreg|mask|z,xmmrm128|b64,imm8 AVX512VL	
VGETMANTPD	Xmmreg | mask | z， xmmrm128 | b64， imm8 AVX512VL	
VGETMANTPD	ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL	
VGETMANTPD	Ymmreg | mask | z， ymmrm256 | b64， imm8 AVX512VL	
VGETMANTPD	zmmreg|mask|z,zmmrm512|b64|sae,imm8 AVX512	
VGETMANTPD	Zmmreg | mask | z， zmmrm512 | b64 | sae， imm8 AVX512	
VGETMANTPS	xmmreg|mask|z,xmmrm128|b32,imm8 AVX512VL	
VGETMANTPS	Xmmreg | mask | z， xmmrm128 | b32， imm8 AVX512VL	
VGETMANTPS	ymmreg|mask|z,ymmrm256|b32,imm8 AVX512VL	
VGETMANTPS	Ymmreg | mask | z， ymmrm256 | b32， imm8 AVX512VL	
VGETMANTPS	zmmreg|mask|z,zmmrm512|b32|sae,imm8 AVX512	
VGETMANTPS	Zmmreg | mask | z， zmmrm512 | b32 | sae， imm8 AVX512	
	xmmreg|mask|z,xmmreg,xmmrm64|sae,imm8 AVX512	
VGETMANTSD	Xmmreg | mask | z， xmmreg， xmmrm64 | sae， imm8	
VGETMANTSD	AVX512	
VGETMANTSS	xmmreg|mask|z,xmmreg,xmmrm32|sae,imm8 AVX512	

VGETMANTSS	Xmmreg | mask | z，xmmreg，xmmrm32 | sae，imm8
	AVX512
	ymmreg|mask|z,ymmreg*,xmmrm128,imm8 AVX512VL
VINSERTF32X4	Ymmreg | mask | z，ymmreg *， xmmrm128，imm8
VINSERTF32X4	AVX512VL
VINSERTF32X4	zmmreg|mask|z,zmmreg*,xmmrm128,imm8 AVX512
VINSERTF32X4	Zmmreg | mask | z，zmmreg *， xmmrm128，imm8 AVX512
	zmmreg|mask|z,zmmreg*,ymmrm256,imm8 AVX512DQ
VINSERTF32X8	Zmmreg | mask | z，zmmreg *， ymmrm256，imm8
VINSERTF32X8	AVX512DQ
	ymmreg|mask|z,ymmreg*,xmmrm128,imm8 AVX512VL/DQ
VINSERTF64X2	Ymmreg | mask | z，ymmreg *， xmmrm128，imm8
VINSERTF64X2	AVX512VL/DQ
	zmmreg|mask|z,zmmreg*,xmmrm128,imm8 AVX512DQ
VINSERTF64X2	Zmmreg | mask | z，zmmreg *， xmmrm128，imm8
VINSERTF64X2	AVX512DQ

209

VINSERTF64X4	zmmreg|mask|z,zmmreg*,ymmrm256,imm8 AVX512
VINSERTF64X4	Zmmreg | mask | z， zmmreg * ， ymmrm256， imm8 AVX512
	ymmreg|mask|z,ymmreg*,xmmrm128,imm8 AVX512VL
VINSERTI32X4	Ymmreg | mask | z， ymmreg * ， xmmrm128， imm8
VINSERTI32X4	AVX512VL
VINSERTI32X4	zmmreg|mask|z,zmmreg*,xmmrm128,imm8 AVX512
VINSERTI32X4	Zmmreg | mask | z， zmmreg * ， xmmrm128， imm8 AVX512
	zmmreg|mask|z,zmmreg*,ymmrm256,imm8 AVX512DQ
VINSERTI32X8	Zmmreg | mask | z， zmmreg * ， ymmrm256， imm8
VINSERTI32X8	AVX512DQ
	ymmreg|mask|z,ymmreg*,xmmrm128,imm8 AVX512VL/DQ
VINSERTI64X2	Ymmreg | mask | z， ymmreg * ， xmmrm128， imm8
VINSERTI64X2	AVX512VL/DQ
	zmmreg|mask|z,zmmreg*,xmmrm128,imm8 AVX512DQ
VINSERTI64X2	Zmmreg | mask | z， zmmreg * ， xmmrm128， imm8
VINSERTI64X2	AVX512DQ
VINSERTI64X4	zmmreg|mask|z,zmmreg*,ymmrm256,imm8 AVX512
VINSERTI64X4	Zmmreg | mask | z， zmmreg * ， ymmrm256， imm8 AVX512
VINSERTPS	xmmreg,xmmreg*,xmmrm32,imm8 AVX512
VINSERTPS	Xmmreg， xmmreg * ， xmmrm32， imm8 AVX512
VMAXPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VMAXPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VMAXPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VMAXPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|sae AVX512
VMAXPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | sae
VMAXPD	AVX512
VMAXPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VMAXPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VMAXPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VMAXPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|sae AVX512
VMAXPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | sae
VMAXPS	AVX512
VMAXSD	xmmreg|mask|z,xmmreg*,xmmrm64|sae AVX512
VMAXSD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | sae AVX512
VMAXSS	xmmreg|mask|z,xmmreg*,xmmrm32|sae AVX512
VMAXSS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | sae AVX512
VMINPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VMINPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VMINPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VMINPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|sae AVX512
VMINPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | sae
VMINPD	AVX512
VMINPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VMINPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VMINPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VMINPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VMINPS	zmmreg|mask|z,zmmreg*,zmmrm512|b32|sae AVX512

VMINPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | sae	
	AVX512	
VMINSD	xmmreg|mask|z,xmmreg*,xmmrm64|sae AVX512	
VMINSD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | sae AVX512	
VMINSS	xmmreg|mask|z,xmmreg*,xmmrm32|sae AVX512	
VMINSS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | sae AVX512	
	xmmreg|mask|z,xmmrm128	
VMOVAPD	Xmmreg | mask | z，	AVX512VL
VMOVAPD	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVAPD	Ymmreg | mask | z，	AVX512VL
VMOVAPD	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVAPD	Zmmreg | mask | z，	AVX512
VMOVAPD	zmmrm512	AVX512
VMOVAPD	xmmreg|mask|z,xmmreg	AVX512VL
VMOVAPD	| mask | z， xmmreg	AVX512VL
VMOVAPD	ymmreg|mask|z,ymmreg	AVX512VL
VMOVAPD	Ymmreg | mask | z， ymmreg	AVX512VL
VMOVAPD	zmmreg|mask|z,zmmreg	AVX512
VMOVAPD	Zmmreg | mask | z， zmmreg	AVX512
VMOVAPD	mem128|mask,xmmreg	AVX512VL
VMOVAPD	Mem128 | mask， xmmreg	AVX512VL
VMOVAPD	mem256|mask,ymmreg	AVX512VL
VMOVAPD	Mem256 | 面具， ymmreg	AVX512VL
VMOVAPD	mem512|mask,zmmreg	AVX512
VMOVAPD	Mem512 | mask， zmmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVAPS	Xmmreg | mask | z，	AVX512VL
VMOVAPS	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVAPS	Ymmreg | mask | z，	AVX512VL
VMOVAPS	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVAPS	Zmmreg | mask | z，	AVX512
VMOVAPS	zmmrm512	AVX512
VMOVAPS	xmmreg|mask|z,xmmreg	AVX512VL
VMOVAPS	| mask | z， xmmreg	AVX512VL
VMOVAPS	ymmreg|mask|z,ymmreg	AVX512VL
VMOVAPS	Ymmreg | mask | z， ymmreg	AVX512VL
VMOVAPS	zmmreg|mask|z,zmmreg	AVX512
VMOVAPS	Zmmreg | mask | z， zmmreg	AVX512
VMOVAPS	mem128|mask,xmmreg	AVX512VL
VMOVAPS	Mem128 | mask， xmmreg	AVX512VL
VMOVAPS	mem256|mask,ymmreg	AVX512VL
VMOVAPS	Mem256 | 面具， ymmreg	AVX512VL
VMOVAPS	mem512|mask,zmmreg	AVX512
VMOVAPS	Mem512 | mask， zmmreg	AVX512
VMOVD	xmmreg,rm32	AVX512
VMOVD	Xmmreg， rm32	AVX512

VMOVD	rm32,xmmreg	AVX512
VMOVD	Rm32，xmmreg	AVX512
	xmmreg|mask|z,xmmrm64	
VMOVDDUP	Xmmreg | mask | z，	AVX512VL
VMOVDDUP	xmmrm64	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVDDUP	Ymmreg | mask | z，	AVX512VL
VMOVDDUP	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVDDUP	Zmmreg | mask | z，	AVX512
VMOVDDUP	zmmrm512	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVDQA32	Xmmreg | mask | z，	AVX512VL
VMOVDQA32	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVDQA32	Ymmreg | mask | z，	AVX512VL
VMOVDQA32	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVDQA32	Zmmreg | mask | z，	AVX512
VMOVDQA32	zmmrm512	AVX512
	xmmrm128|mask|z,xmmreg	
VMOVDQA32	Xmmrm128 | mask | z，	AVX512VL
VMOVDQA32	xmmreg	AVX512VL
	ymmrm256|mask|z,ymmreg	
VMOVDQA32	Ymmrm256 | mask | z，	AVX512VL
VMOVDQA32	ymmrreg	AVX512VL
VMOVDQA32	zmmrm512|mask|z,zmmreg	AVX512
VMOVDQA32	Zmmrm512 | mask | z，zmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVDQA64	Xmmreg | mask | z，	AVX512VL
VMOVDQA64	xmmrm128	AVX512VL

210

	ymmreg|mask|z,ymmrm256	
VMOVDQA64	Ymmreg | mask | z，	AVX512VL
VMOVDQA64	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVDQA64	Zmmreg | mask | z，	AVX512
VMOVDQA64	zmmrm512	AVX512
	xmmrm128|mask|z,xmmreg	
VMOVDQA64	Xmmrm128 | mask | z，	AVX512VL
VMOVDQA64	xmmreg	AVX512VL
	ymmrm256|mask|z,ymmreg	
VMOVDQA64	Ymmrm256 | mask | z，	AVX512VL
VMOVDQA64	ymmrreg	AVX512VL
VMOVDQA64	zmmrm512|mask|z,zmmreg	AVX512
VMOVDQA64	Zmmrm512 | mask | z， zmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVDQU16	Xmmreg | mask | z，	AVX512VL/BW
VMOVDQU16	xmmrm128	AVX512VL/BW
	ymmreg|mask|z,ymmrm256	
VMOVDQU16	Ymmreg | mask | z，	AVX512VL/BW
VMOVDQU16	ymmrm256	AVX512VL/BW
	zmmreg|mask|z,zmmrm512	
VMOVDQU16	Zmmreg | mask | z，	AVX512BW
VMOVDQU16	zmmrm512	AVX512BW
	xmmrm128|mask|z,xmmreg	
VMOVDQU16	Xmmrm128 | mask | z，	AVX512VL/BW
VMOVDQU16	xmmreg	AVX512VL/BW
	ymmrm256|mask|z,ymmreg	
VMOVDQU16	Ymmrm256 | mask | z，	AVX512VL/BW
VMOVDQU16	ymmrreg	AVX512VL/BW
VMOVDQU16	zmmrm512|mask|z,zmmreg	AVX512BW
VMOVDQU16	Zmmrm512 | mask | z， zmreg	AVX512BW
	xmmreg|mask|z,xmmrm128	
VMOVDQU32	Xmmreg | mask | z，	AVX512VL
VMOVDQU32	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVDQU32	Ymmreg | mask | z，	AVX512VL
VMOVDQU32	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVDQU32	Zmmreg | mask | z，	AVX512
VMOVDQU32	zmmrm512	AVX512
	xmmrm128|mask|z,xmmreg	
VMOVDQU32	Xmmrm128 | mask | z，	AVX512VL
VMOVDQU32	xmmreg	AVX512VL
	ymmrm256|mask|z,ymmreg	
VMOVDQU32	Ymmrm256 | mask | z，	AVX512VL
VMOVDQU32	ymmrreg	AVX512VL
VMOVDQU32	zmmrm512|mask|z,zmmreg	AVX512
VMOVDQU32	Zmmrm512 | mask | z， zmreg	AVX512
VMOVDQU64	xmmreg|mask|z,xmmrm128	AVX512VL
VMOVDQU64	Xmmreg | mask | z，	AVX512VL

	xmmrm128	
	ymmreg|mask|z,ymmrm256	
VMOVDQU64	Ymmreg | mask | z，	AVX512VL
VMOVDQU64	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVDQU64	Zmmreg | mask | z，	AVX512
VMOVDQU64	zmmrm512	AVX512
	xmmrm128|mask|z,xmmreg	
VMOVDQU64	Xmmrm128 | mask | z，	AVX512VL
VMOVDQU64	xmmreg	AVX512VL
	ymmrm256|mask|z,ymmreg	
VMOVDQU64	Ymmrm256 | mask | z，	AVX512VL
VMOVDQU64	ymmrreg	AVX512VL
VMOVDQU64	zmmrm512|mask|z,zmmreg	AVX512
VMOVDQU64	Zmmrm512 | mask | z， zmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVDQU8	Xmmreg | mask | z，	AVX512VL/BW
VMOVDQU8	xmmrm128	AVX512VL/BW
	ymmreg|mask|z,ymmrm256	
VMOVDQU8	Ymmreg | mask | z，	AVX512VL/BW
VMOVDQU8	ymmrm256	AVX512VL/BW
	zmmreg|mask|z,zmmrm512	
VMOVDQU8	Zmmreg | mask | z，	AVX512BW
VMOVDQU8	zmmrm512	AVX512BW
	xmmrm128|mask|z,xmmreg	
VMOVDQU8	Xmmrm128 | mask | z，	AVX512VL/BW
VMOVDQU8	xmmreg	AVX512VL/BW
	ymmrm256|mask|z,ymmreg	
VMOVDQU8	Ymmrm256 | mask | z，	AVX512VL/BW
VMOVDQU8	ymmrreg	AVX512VL/BW
VMOVDQU8	zmmrm512|mask|z,zmmreg	AVX512BW
VMOVDQU8	Zmmrm512 | mask | z， zmreg	AVX512BW
	xmmreg,xmmreg*,xmmreg	
VMOVHLPS	Xmmreg， xmmreg * ，	AVX512
VMOVHLPS	xmmreg	AVX512
	xmmreg,xmmreg*,mem64	
VMOVHPD	Xmmreg， xmmreg * ，	AVX512
VMOVHPD	mem64	AVX512
VMOVHPD	mem64,xmmreg	AVX512
VMOVHPD	Mem64， xmmreg	AVX512
	xmmreg,xmmreg*,mem64	
VMOVHPS	Xmmreg， xmmreg * ，	AVX512
VMOVHPS	mem64	AVX512
VMOVHPS	mem64,xmmreg	AVX512
VMOVHPS	Mem64， xmmreg	AVX512
	xmmreg,xmmreg*,xmmreg	
VMOVLHPS	Xmmreg， xmmreg * ，	AVX512
VMOVLHPS	xmmreg	AVX512
	xmmreg,xmmreg*,mem64	
VMOVLPD	Xmmreg， xmmreg * ，	AVX512
VMOVLPD	mem64	AVX512

VMOVLPD	mem64,xmmreg	AVX512
VMOVLPD	Mem64，xmmreg	AVX512
	xmmreg,xmmreg*,mem64	
VMOVLPS	Xmmreg，xmmreg * ，	AVX512
VMOVLPS	mem64	AVX512
VMOVLPS	mem64,xmmreg	AVX512
VMOVLPS	Mem64，xmmreg	AVX512
VMOVNTDQ	mem128,xmmreg	AVX512VL
VMOVNTDQ	Mem128，xmmreg	AVX512VL
VMOVNTDQ	mem256,ymmreg	AVX512VL
VMOVNTDQ	Mem256，ymmreg	AVX512VL
VMOVNTDQ	mem512,zmmreg	AVX512
VMOVNTDQ	Mem512，zmmreg	AVX512
VMOVNTDQA	xmmreg,mem128	AVX512VL
VMOVNTDQA	Xmmreg，mem128	AVX512VL
VMOVNTDQA	ymmreg,mem256	AVX512VL
VMOVNTDQA	Ymmreg，mem256	AVX512VL
VMOVNTDQA	zmmreg,mem512	AVX512
VMOVNTDQA	Zmmreg，mem512	AVX512
VMOVNTPD	mem128,xmmreg	AVX512VL
VMOVNTPD	Mem128，xmmreg	AVX512VL
VMOVNTPD	mem256,ymmreg	AVX512VL
VMOVNTPD	Mem256，ymmreg	AVX512VL
VMOVNTPD	mem512,zmmreg	AVX512
VMOVNTPD	Mem512，zmmreg	AVX512
VMOVNTPS	mem128,xmmreg	AVX512VL
VMOVNTPS	Mem128，xmmreg	AVX512VL
VMOVNTPS	mem256,ymmreg	AVX512VL
VMOVNTPS	Mem256，ymmreg	AVX512VL
VMOVNTPS	mem512,zmmreg	AVX512
VMOVNTPS	Mem512，zmmreg	AVX512
VMOVQ	xmmreg,rm64	AVX512
VMOVQ	Xmmreg，rm64	AVX512
VMOVQ	rm64,xmmreg	AVX512
VMOVQ	Rm64，xmmreg	AVX512
VMOVQ	xmmreg,xmmrm64	AVX512
VMOVQ	Xmmreg，xmmrm64	AVX512

211

VMOVQ	xmmrm64,xmmreg	AVX512
VMOVQ	Xmmrm64，xmreg	AVX512
VMOVSD	xmmreg|mask|z,mem64	AVX512
VMOVSD	Xmmreg | mask | z，mem64	AVX512
VMOVSD	mem64|mask,xmmreg	AVX512
VMOVSD	Mem64 | mask，xmmreg	AVX512
VMOVSD	xmmreg|mask|z,xmmreg*,xmmreg	AVX512
VMOVSD	Xmmreg | mask | z，xmmreg *，xmmreg	AVX512
VMOVSD	xmmreg|mask|z,xmmreg*,xmmreg	AVX512
VMOVSD	Xmmreg | mask | z，xmmreg *，xmmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVSHDUP	Xmmreg | mask | z，	AVX512VL
VMOVSHDUP	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVSHDUP	Ymmreg | mask | z，	AVX512VL
VMOVSHDUP	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVSHDUP	Zmmreg | mask | z，	AVX512
VMOVSHDUP	zmmrm512	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVSLDUP	Xmmreg | mask | z，	AVX512VL
VMOVSLDUP	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVSLDUP	Ymmreg | mask | z，	AVX512VL
VMOVSLDUP	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVSLDUP	Zmmreg | mask | z，	AVX512
VMOVSLDUP	zmmrm512	AVX512
VMOVSS		
VMOVSS	xmmreg|mask|z,mem32	AVX512
VMOVSS	Xmmreg | mask | z，mem32	AVX512
VMOVSS		
VMOVSS	mem32|mask,xmmreg	AVX512
VMOVSS	Mem32 | mask，xmmreg	AVX512
VMOVSS		
VMOVSS	xmmreg|mask|z,xmmreg*,xmmreg	AVX512
VMOVSS	Xmmreg | mask | z，xmmreg *，xmmreg	AVX512
VMOVSS		
VMOVSS	xmmreg|mask|z,xmmreg*,xmmreg	AVX512
VMOVSS	Xmmreg | mask | z，xmmreg *，xmmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVUPD	Xmmreg | mask | z，	AVX512VL
VMOVUPD	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVUPD	Ymmreg | mask | z，	AVX512VL
VMOVUPD	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVUPD	Zmmreg | mask | z，	AVX512
VMOVUPD	zmmrm512	AVX512
VMOVUPD	xmmreg|mask|z,xmmreg	AVX512VL
VMOVUPD	| mask | z，xmmreg	AVX512VL

VMOVUPD	ymmreg|mask|z,ymmreg	AVX512VL
VMOVUPD	Ymmreg | mask | z， ymmreg	AVX512VL
VMOVUPD	zmmreg|mask|z,zmmreg	AVX512
VMOVUPD	Zmmreg | mask | z， zmmreg	AVX512
VMOVUPD	mem128|mask,xmmreg	AVX512VL
VMOVUPD	Mem128 | mask， xmmreg	AVX512VL
VMOVUPD	mem256|mask,ymmreg	AVX512VL
VMOVUPD	Mem256 | 面具， ymmreg	AVX512VL
VMOVUPD	mem512|mask,zmmreg	AVX512
VMOVUPD	Mem512 | mask， zmmreg	AVX512
	xmmreg|mask|z,xmmrm128	
VMOVUPS	Xmmreg | mask | z，	AVX512VL
VMOVUPS	xmmrm128	AVX512VL
	ymmreg|mask|z,ymmrm256	
VMOVUPS	Ymmreg | mask | z，	AVX512VL
VMOVUPS	ymmrm256	AVX512VL
	zmmreg|mask|z,zmmrm512	
VMOVUPS	Zmmreg | mask | z，	AVX512
VMOVUPS	zmmrm512	AVX512
VMOVUPS	xmmreg|mask|z,xmmreg	AVX512VL
VMOVUPS	| mask | z， xmmreg	AVX512VL
VMOVUPS	ymmreg|mask|z,ymmreg	AVX512VL
VMOVUPS	Ymmreg | mask | z， ymmreg	AVX512VL
VMOVUPS	zmmreg|mask|z,zmmreg	AVX512
VMOVUPS	Zmmreg | mask | z， zmmreg	AVX512
VMOVUPS	mem128|mask,xmmreg	AVX512VL
VMOVUPS	Mem128 | mask， xmmreg	AVX512VL
VMOVUPS	mem256|mask,ymmreg	AVX512VL
VMOVUPS	Mem256 | 面具， ymmreg	AVX512VL
VMOVUPS	mem512|mask,zmmreg	AVX512
VMOVUPS	Mem512 | mask， zmmreg	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VMULPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64	
VMULPD	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VMULPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64	
VMULPD	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|er AVX512	
VMULPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | er	
VMULPD	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL	
VMULPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32	
VMULPS	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL	
VMULPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32	
VMULPS	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|er AVX512	
VMULPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | er	
VMULPS	AVX512	
VMULSD		
VMULSD	xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512	

	Xmmreg | mask | z，xmmreg *，xmmrm64 | er AVX512	
VMULSS	xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512	
VMULSS	Xmmreg | mask | z，xmmreg *，xmmrm32 | er AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/DQ	
VORPD	Xmmreg | mask | z，xmmreg *，xmmrm128 | b64	
VORPD	AVX512VL/DQ	
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/DQ	
VORPD	Ymmreg | mask | z，ymmreg *，ymmrm256 | b64	
VORPD	AVX512VL/DQ	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512DQ	
VORPD	Zmmreg | mask | z，zmmreg *，zmmrm512 | b64	
VORPD	AVX512DQ	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/DQ	
VORPS	Xmmreg | mask | z，xmmreg *，xmmrm128 | b32	
VORPS	AVX512VL/DQ	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/DQ	
VORPS	Ymmreg | mask | z，ymmreg *，ymmrm256 | b32	
VORPS	AVX512VL/DQ	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512DQ	
VORPS	Zmmreg | mask | z，zmmreg *，zmmrm512 | b32	
VORPS	AVX512DQ	
	xmmreg|mask|z,xmmrm128	
VPABSB	Xmmreg | mask | z，	AVX512VL/BW
VPABSB	xmmrm128	AVX512VL/BW
	ymmreg|mask|z,ymmrm256	
VPABSB	Ymmreg | mask | z，	AVX512VL/BW
VPABSB	ymmrm256	AVX512VL/BW
	zmmreg|mask|z,zmmrm512	
VPABSB	Zmmreg | mask | z，	AVX512BW
VPABSB	zmmrm512	AVX512BW
VPABSD	xmmreg|mask|z,xmmrm128|b32 AVX512VL	
VPABSD	Xmmreg | mask | z，xmmrm128 | b32 AVX512VL	
VPABSD	ymmreg|mask|z,ymmrm256|b32 AVX512VL	
VPABSD	Ymmreg | mask | z，ymmrm256 | b32 AVX512VL	
VPABSD	zmmreg|mask|z,zmmrm512|b32 AVX512	
VPABSD	Zmmrreg | mask | z，zmmrm512 | b32 AVX512	
VPABSQ	xmmreg|mask|z,xmmrm128|b64 AVX512VL	
VPABSQ	Xmmreg | mask | z，xmmrm128 | b64 AVX512VL	

212

VPABSQ	ymmreg|mask|z,ymmrm256|b64 AVX512VL
VPABSQ	Ymmreg | mask | z， ymmrm256 | b64 AVX512VL
VPABSQ	zmmreg|mask|z,zmmrm512|b64 AVX512
VPABSQ	Zmmrreg | mask | z， zmmrm512 | b64 AVX512
	xmmreg|mask|z,xmmrm128
VPABSW	Xmmreg | mask | z， AVX512VL/BW
VPABSW	xmmrm128 AVX512VL/BW
	ymmreg|mask|z,ymmrm256
VPABSW	Ymmreg | mask | z， AVX512VL/BW
VPABSW	ymmrm256 AVX512VL/BW
	zmmreg|mask|z,zmmrm512
VPABSW	Zmmreg | mask | z， AVX512BW
VPABSW	zmmrm512 AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/BW
VPACKSSDW	Xmmreg | mask | z， xmmreg *， xmmrm128 | b32
VPACKSSDW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/BW
VPACKSSDW	Ymmreg | mask | z， ymmreg *， ymmrm256 | b32
VPACKSSDW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512BW
VPACKSSDW	Zmmreg | mask | z， zmmreg *， zmmrm512 | b32
VPACKSSDW	AVX512BW
VPACKSSWB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPACKSSWB	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPACKSSWB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPACKSSWB	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPACKSSWB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPACKSSWB	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/BW
VPACKUSDW	Xmmreg | mask | z， xmmreg *， xmmrm128 | b32
VPACKUSDW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/BW
VPACKUSDW	Ymmreg | mask | z， ymmreg *， ymmrm256 | b32
VPACKUSDW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512BW
VPACKUSDW	Zmmreg | mask | z， zmmreg *， zmmrm512 | b32
VPACKUSDW	AVX512BW
VPACKUSWB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPACKUSWB	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPACKUSWB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPACKUSWB	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPACKUSWB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPACKUSWB	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
VPADDB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDB	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPADDB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDB	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPADDB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDB	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
VPADDD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL

VPADDD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPADDD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPADDD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VPADDD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPADDD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VPADDQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPADDQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VPADDQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPADDQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
VPADDQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPADDQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VPADDSB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDSB	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPADDSB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDSB	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPADDSB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDSB	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPADDSW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDSW	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPADDSW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDSW	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPADDSW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDSW	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPADDUSB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDUSB	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPADDUSB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDUSB	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPADDUSB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDUSB	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPADDUSW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDUSW	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPADDUSW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDUSW	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPADDUSW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDUSW	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPADDW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPADDW	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPADDW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPADDW	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPADDW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPADDW	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128,imm8 AVX512VL/BW
VPALIGNR	Xmmreg | mask | z， xmmreg * ， xmmrm128， imm8
VPALIGNR	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256,imm8 AVX512VL/BW
VPALIGNR	Ymmreg | mask | z， ymmreg * ， ymmrm256， imm8
VPALIGNR	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512,imm8 AVX512BW
VPALIGNR	Zmmreg | mask | z， zmmreg * ， zmmrm512， imm8
VPALIGNR	AVX512BW

VPANDD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPANDD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPANDD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPANDD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VPANDD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPANDD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VPANDND	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPANDND	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPANDND	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPANDND	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VPANDND	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPANDND	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VPANDNQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPANDNQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VPANDNQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPANDNQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
VPANDNQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPANDNQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VPANDQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPANDQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL

213

	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VPANDQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64	
VPANDQ	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512	
VPANDQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64	
VPANDQ	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPAVGB	Xmmreg | mask | z， xmmreg * ， xmmrm128	
VPAVGB	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPAVGB	Ymmreg | mask | z， ymmreg * ， ymmrm256	
VPAVGB	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPAVGB	Zmmreg | mask | z， zmmreg * ， zmmrm512	
VPAVGB	AVX512BW	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPAVGW	Xmmreg | mask | z， xmmreg * ， xmmrm128	
VPAVGW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPAVGW	Ymmreg | mask | z， ymmreg * ， ymmrm256	
VPAVGW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPAVGW	Zmmreg | mask | z， zmmreg * ， zmmrm512	
VPAVGW	AVX512BW	
	xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/BW	
VPBLENDMB	Xmmreg | mask | z， xmmreg， xmmrm128	
VPBLENDMB	AVX512VL/BW	
	ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/BW	
VPBLENDMB	Ymmreg | mask | z， ymmreg， ymmrm256	
VPBLENDMB	AVX512VL/BW	
VPBLENDMB	zmmreg|mask|z,zmmreg,zmmrm512 AVX512BW	
VPBLENDMB	Zmmreg | mask | z， zmmreg， zmmrm512 AVX512BW	
	xmmreg|mask|z,xmmreg,xmmrm128|b32	
VPBLENDMD	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPBLENDMD	| b32	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b32	
VPBLENDMD	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPBLENDMD	| b32	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b32	
VPBLENDMD	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPBLENDMD	| b32	AVX512
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VPBLENDMQ	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPBLENDMQ	| b64	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VPBLENDMQ	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPBLENDMQ	| b64	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VPBLENDMQ	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPBLENDMQ	| b64	AVX512

	xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/BW	
VPBLENDMW	Xmmreg | mask | z， xmmreg， xmmrm128	
VPBLENDMW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/BW	
VPBLENDMW	Ymmreg | mask | z， ymmreg， ymmrm256	
VPBLENDMW	AVX512VL/BW	
VPBLENDMW	zmmreg|mask|z,zmmreg,zmmrm512 AVX512BW	
VPBLENDMW	Zmmreg | mask | z， zmmreg， zmmrm512 AVX512BW	
	xmmreg|mask|z,xmmrm8	
VPBROADCASTB	Xmmreg | mask | z，	AVX512VL/BW
VPBROADCASTB	xmmrm8	AVX512VL/BW
	ymmreg|mask|z,xmmrm8	
VPBROADCASTB	Ymmreg | mask | z，	AVX512VL/BW
VPBROADCASTB	xmmrm8	AVX512VL/BW
	zmmreg|mask|z,xmmrm8	
VPBROADCASTB	Zmmreg | mask | z，	AVX512BW
VPBROADCASTB	xmmrm8	AVX512BW
VPBROADCASTB	xmmreg|mask|z,reg8	AVX512VL/BW
VPBROADCASTB	Xmmreg | mask | z， reg8	AVX512VL/BW
VPBROADCASTB	xmmreg|mask|z,reg16	AVX512VL/BW
VPBROADCASTB	Xmmreg | mask | z， reg16	AVX512VL/BW
VPBROADCASTB	xmmreg|mask|z,reg32	AVX512VL/BW
VPBROADCASTB	Xmmreg | mask | z， reg32	AVX512VL/BW
VPBROADCASTB	xmmreg|mask|z,reg64	AVX512VL/BW
VPBROADCASTB	Xmmreg | mask | z， reg64	AVX512VL/BW
VPBROADCASTB	ymmreg|mask|z,reg8	AVX512VL/BW
VPBROADCASTB	Ymmreg | mask | z， reg8	AVX512VL/BW
VPBROADCASTB	ymmreg|mask|z,reg16	AVX512VL/BW
VPBROADCASTB	Ymmreg | mask | z， reg16	AVX512VL/BW
VPBROADCASTB	ymmreg|mask|z,reg32	AVX512VL/BW
VPBROADCASTB	Ymmreg | mask | z， reg32	AVX512VL/BW
VPBROADCASTB	ymmreg|mask|z,reg64	AVX512VL/BW
VPBROADCASTB	Ymmreg | mask | z， reg64	AVX512VL/BW
VPBROADCASTB	zmmreg|mask|z,reg8	AVX512BW
VPBROADCASTB	Zmmreg | mask | z， reg8	AVX512BW
VPBROADCASTB	zmmreg|mask|z,reg16	AVX512BW
VPBROADCASTB	Zmmreg | mask | z， reg16	AVX512BW
VPBROADCASTB	zmmreg|mask|z,reg32	AVX512BW
VPBROADCASTB	Zmmreg | mask | z， reg32	AVX512BW
VPBROADCASTB	zmmreg|mask|z,reg64	AVX512BW
VPBROADCASTB	Zmmreg | mask | z， reg64	AVX512BW
VPBROADCASTD	xmmreg|mask|z,mem32	AVX512VL
VPBROADCASTD	Xmmreg | mask | z， mem32	AVX512VL
VPBROADCASTD	ymmreg|mask|z,mem32	AVX512VL
VPBROADCASTD	Ymmreg | mask | z， mem32	AVX512VL
VPBROADCASTD	zmmreg|mask|z,mem32	AVX512
VPBROADCASTD	Zmmreg | mask | z， mem32	AVX512
VPBROADCASTD	xmmreg|mask|z,xmmreg	AVX512VL
VPBROADCASTD	| mask | z， xmmreg	AVX512VL
VPBROADCASTD	ymmreg|mask|z,xmmreg	AVX512VL

VPBROADCASTD	Ymmreg | mask | z， xmreg	AVX512VL
VPBROADCASTD	zmmreg|mask|z,xmmreg	AVX512
VPBROADCASTD	Zmmreg | mask | z， xmmreg	AVX512
VPBROADCASTD	xmmreg|mask|z,reg32	AVX512VL
VPBROADCASTD	Xmmreg | mask | z， reg32	AVX512VL
VPBROADCASTD	ymmreg|mask|z,reg32	AVX512VL
VPBROADCASTD	Ymmreg | mask | z， reg32	AVX512VL
VPBROADCASTD	zmmreg|mask|z,reg32	AVX512
VPBROADCASTD	Zmmreg | mask | z， reg32	AVX512
VPBROADCASTMB2Q	xmmreg,kreg	AVX512VL/CD
VPBROADCASTMB2Q	Xmmreg， kreg	AVX512VL/CD
VPBROADCASTMB2Q	ymmreg,kreg	AVX512VL/CD
VPBROADCASTMB2Q	Ymmreg， kreg	AVX512VL/CD
VPBROADCASTMB2Q	zmmreg,kreg	AVX512CD
VPBROADCASTMB2Q	Zmmreg， kreg	AVX512CD
VPBROADCASTMW2D		
VPBROADCASTMW2	xmmreg,kreg	AVX512VL/CD
D	Xmmreg， kreg	AVX512VL/CD
VPBROADCASTMW2D		
VPBROADCASTMW2	ymmreg,kreg	AVX512VL/CD
D	Ymmreg， kreg	AVX512VL/CD
VPBROADCASTMW2D		
VPBROADCASTMW2	zmmreg,kreg	AVX512CD
D	Zmmreg， kreg	AVX512CD
VPBROADCASTQ	xmmreg|mask|z,mem64	AVX512VL
VPBROADCASTQ	Xmmreg | mask | z， mem64	AVX512VL
VPBROADCASTQ	ymmreg|mask|z,mem64	AVX512VL
VPBROADCASTQ	Ymmreg | mask | z， mem64	AVX512VL
VPBROADCASTQ	zmmreg|mask|z,mem64	AVX512
VPBROADCASTQ	Zmmreg | mask | z， mem64	AVX512
VPBROADCASTQ	xmmreg|mask|z,xmmreg	AVX512VL
VPBROADCASTQ	| mask | z， xmmreg	AVX512VL

214

VPBROADCASTQ	ymmreg|mask|z,xmmreg	AVX512VL
VPBROADCASTQ	Ymmreg | mask | z，xmreg	AVX512VL
VPBROADCASTQ	zmmreg|mask|z,xmmreg	AVX512
VPBROADCASTQ	Zmmreg | mask | z，xmmreg	AVX512
VPBROADCASTQ	xmmreg|mask|z,reg64	AVX512VL
VPBROADCASTQ	Xmmreg | mask | z，reg64	AVX512VL
VPBROADCASTQ	ymmreg|mask|z,reg64	AVX512VL
VPBROADCASTQ	Ymmreg | mask | z，reg64	AVX512VL
VPBROADCASTQ	zmmreg|mask|z,reg64	AVX512
VPBROADCASTQ	Zmmreg | mask | z，reg64	AVX512
	xmmreg|mask|z,xmmrm16	
VPBROADCASTW	Xmmrreg | mask | z，	AVX512VL/BW
VPBROADCASTW	xmmrm16	AVX512VL/BW
	ymmreg|mask|z,xmmrm16	
VPBROADCASTW	Ymmreg | mask | z，	AVX512VL/BW
VPBROADCASTW	xmmrm16	AVX512VL/BW
	zmmreg|mask|z,xmmrm16	
VPBROADCASTW	Zmmreg | mask | z，	AVX512BW
VPBROADCASTW	xmmrm16	AVX512BW
VPBROADCASTW	xmmreg|mask|z,reg16	AVX512VL/BW
VPBROADCASTW	Xmmreg | mask | z，reg16	AVX512VL/BW
VPBROADCASTW	xmmreg|mask|z,reg32	AVX512VL/BW
VPBROADCASTW	Xmmreg | mask | z，reg32	AVX512VL/BW
VPBROADCASTW	xmmreg|mask|z,reg64	AVX512VL/BW
VPBROADCASTW	Xmmreg | mask | z，reg64	AVX512VL/BW
VPBROADCASTW	ymmreg|mask|z,reg16	AVX512VL/BW
VPBROADCASTW	Ymmreg | mask | z，reg16	AVX512VL/BW
VPBROADCASTW	ymmreg|mask|z,reg32	AVX512VL/BW
VPBROADCASTW	Ymmreg | mask | z，reg32	AVX512VL/BW
VPBROADCASTW	ymmreg|mask|z,reg64	AVX512VL/BW
VPBROADCASTW	Ymmreg | mask | z，reg64	AVX512VL/BW
VPBROADCASTW	zmmreg|mask|z,reg16	AVX512BW
VPBROADCASTW	Zmmreg | mask | z，reg16	AVX512BW
VPBROADCASTW	zmmreg|mask|z,reg32	AVX512BW
VPBROADCASTW	Zmmreg | mask | z，reg32	AVX512BW
VPBROADCASTW	zmmreg|mask|z,reg64	AVX512BW
VPBROADCASTW	Zmmreg | mask | z，reg64	AVX512BW
	kreg|mask,xmmreg,xmmrm128,imm8 AVX512VL/BW	
VPCMPB	Kreg | mask，xmmreg，xmmrm128，imm8	
VPCMPB	AVX512VL/BW	
	kreg|mask,ymmreg,ymmrm256,imm8 AVX512VL/BW	
VPCMPB	Kreg | mask，ymmreg，ymmrm256，imm8	
VPCMPB	AVX512VL/BW	
VPCMPB	kreg|mask,zmmreg,zmmrm512,imm8 AVX512BW	
VPCMPB	Kreg | mask，zmmreg，zmmrm512，imm8 AVX512BW	
	kreg|mask,xmmreg,xmmrm128|b32,imm8 AVX512VL	
VPCMPD	Kreg | mask，xmmreg，xmmrm128 | b32，imm8	
VPCMPD	AVX512VL	
VPCMPD	kreg|mask,ymmreg,ymmrm256|b32,imm8 AVX512VL	
VPCMPD	Kreg | mask，ymmreg，ymmrm256 | b32，imm8	

	AVX512VL
	kreg|mask,zmmreg,zmmrm512|b32,imm8 AVX512
VPCMPD	Kreg | mask， zmmreg， zmmrm512 | b32， imm8
VPCMPD	AVX512
VPCMPEQB	kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPCMPEQB	克雷格 | 面具， xmmreg， xmmrm128 AVX512VL/BW
VPCMPEQB	kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPCMPEQB	面具， ymmrm256 AVX512VL/BW
VPCMPEQB	kreg|mask,zmmreg,zmmrm512 AVX512BW
VPCMPEQB	面具， zmreg， zmmrm512 AVX512BW
VPCMPEQD	kreg|mask,xmmreg,xmmrm128|b32 AVX512VL
VPCMPEQD	Kreg | mask， xmmreg， xmmrm128 | b32 AVX512VL
VPCMPEQD	kreg|mask,ymmreg,ymmrm256|b32 AVX512VL
VPCMPEQD	面具， ymmreg， ymmrm256 | b32 AVX512VL
VPCMPEQD	kreg|mask,zmmreg,zmmrm512|b32 AVX512
VPCMPEQD	克雷格 | 面具， zmreg， zmmrm512 | b32 AVX512
VPCMPEQQ	kreg|mask,xmmreg,xmmrm128|b64 AVX512VL
VPCMPEQQ	Kreg | mask， xmmreg， xmmrm128 | b64 AVX512VL
VPCMPEQQ	kreg|mask,ymmreg,ymmrm256|b64 AVX512VL
VPCMPEQQ	面具， ymmreg， ymmrm256 | b64 AVX512VL
VPCMPEQQ	kreg|mask,zmmreg,zmmrm512|b64 AVX512
VPCMPEQQ	克雷格 | 面具， zmreg， zmmrm512 | b64 AVX512
VPCMPEQW	kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPCMPEQW	克雷格 | 面具， xmmreg， xmmrm128 AVX512VL/BW
VPCMPEQW	kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPCMPEQW	面具， ymmrm256 AVX512VL/BW
VPCMPEQW	kreg|mask,zmmreg,zmmrm512 AVX512BW
VPCMPEQW	面具， zmreg， zmmrm512 AVX512BW
VPCMPGTB	kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPCMPGTB	克雷格 | 面具， xmmreg， xmmrm128 AVX512VL/BW
VPCMPGTB	kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPCMPGTB	面具， ymmrm256 AVX512VL/BW
VPCMPGTB	kreg|mask,zmmreg,zmmrm512 AVX512BW
VPCMPGTB	面具， zmreg， zmmrm512 AVX512BW
VPCMPGTD	kreg|mask,xmmreg,xmmrm128|b32 AVX512VL
VPCMPGTD	Kreg | mask， xmmreg， xmmrm128 | b32 AVX512VL
VPCMPGTD	kreg|mask,ymmreg,ymmrm256|b32 AVX512VL
VPCMPGTD	面具， ymmreg， ymmrm256 | b32 AVX512VL
VPCMPGTD	kreg|mask,zmmreg,zmmrm512|b32 AVX512
VPCMPGTD	克雷格 | 面具， zmreg， zmmrm512 | b32 AVX512
VPCMPGTQ	kreg|mask,xmmreg,xmmrm128|b64 AVX512VL
VPCMPGTQ	Kreg | mask， xmmreg， xmmrm128 | b64 AVX512VL
VPCMPGTQ	kreg|mask,ymmreg,ymmrm256|b64 AVX512VL
VPCMPGTQ	面具， ymmreg， ymmrm256 | b64 AVX512VL
VPCMPGTQ	kreg|mask,zmmreg,zmmrm512|b64 AVX512
VPCMPGTQ	克雷格 | 面具， zmreg， zmmrm512 | b64 AVX512
VPCMPGTW	kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPCMPGTW	克雷格 | 面具， xmmreg， xmmrm128 AVX512VL/BW
VPCMPGTW	kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPCMPGTW	面具， ymmrm256 AVX512VL/BW

VPCMPGTW	kreg|mask,zmmreg,zmmrm512 AVX512BW
VPCMPGTW	面具，zmreg，zmmrm512 AVX512BW
	kreg|mask,xmmreg,xmmrm128|b64,imm8 AVX512VL
VPCMPQ	Kreg | mask，xmmreg，xmmrm128 | b64，imm8
VPCMPQ	AVX512VL
	kreg|mask,ymmreg,ymmrm256|b64,imm8 AVX512VL
VPCMPQ	Kreg | mask，ymmreg，ymmrm256 | b64，imm8
VPCMPQ	AVX512VL
	kreg|mask,zmmreg,zmmrm512|b64,imm8 AVX512
VPCMPQ	Kreg | mask，zmmreg，zmmrm512 | b64，imm8
VPCMPQ	AVX512
	kreg|mask,xmmreg,xmmrm128,imm8 AVX512VL/BW
VPCMPUB	Kreg | mask，xmmreg，xmmrm128，imm8
VPCMPUB	AVX512VL/BW
	kreg|mask,ymmreg,ymmrm256,imm8 AVX512VL/BW
VPCMPUB	Kreg | mask，ymmreg，ymmrm256，imm8
VPCMPUB	AVX512VL/BW
VPCMPUB	kreg|mask,zmmreg,zmmrm512,imm8 AVX512BW
VPCMPUB	Kreg | mask，zmmreg，zmmrm512，imm8 AVX512BW
	kreg|mask,xmmreg,xmmrm128|b32,imm8 AVX512VL
VPCMPUD	Kreg | mask，xmmreg，xmmrm128 | b32，imm8
VPCMPUD	AVX512VL

215

	kreg|mask,ymmreg,ymmrm256|b32,imm8 AVX512VL	
VPCMPUD	Kreg | mask， ymmreg， ymmrm256 | b32， imm8	
VPCMPUD	AVX512VL	
	kreg|mask,zmmreg,zmmrm512|b32,imm8 AVX512	
VPCMPUD	Kreg | mask， zmmreg， zmmrm512 | b32， imm8	
VPCMPUD	AVX512	
	kreg|mask,xmmreg,xmmrm128|b64,imm8 AVX512VL	
VPCMPUQ	Kreg | mask， xmmreg， xmmrm128 | b64， imm8	
VPCMPUQ	AVX512VL	
	kreg|mask,ymmreg,ymmrm256|b64,imm8 AVX512VL	
VPCMPUQ	Kreg | mask， ymmreg， ymmrm256 | b64， imm8	
VPCMPUQ	AVX512VL	
	kreg|mask,zmmreg,zmmrm512|b64,imm8 AVX512	
VPCMPUQ	Kreg | mask， zmmreg， zmmrm512 | b64， imm8	
VPCMPUQ	AVX512	
	kreg|mask,xmmreg,xmmrm128,imm8 AVX512VL/BW	
VPCMPUW	Kreg | mask， xmmreg， xmmrm128， imm8	
VPCMPUW	AVX512VL/BW	
	kreg|mask,ymmreg,ymmrm256,imm8 AVX512VL/BW	
VPCMPUW	Kreg | mask， ymmreg， ymmrm256， imm8	
VPCMPUW	AVX512VL/BW	
VPCMPUW	kreg|mask,zmmreg,zmmrm512,imm8 AVX512BW	
VPCMPUW	Kreg | mask， zmmreg， zmmrm512， imm8 AVX512BW	
	kreg|mask,xmmreg,xmmrm128,imm8 AVX512VL/BW	
VPCMPW	Kreg | mask， xmmreg， xmmrm128， imm8	
VPCMPW	AVX512VL/BW	
	kreg|mask,ymmreg,ymmrm256,imm8 AVX512VL/BW	
VPCMPW	Kreg | mask， ymmreg， ymmrm256， imm8	
VPCMPW	AVX512VL/BW	
VPCMPW	kreg|mask,zmmreg,zmmrm512,imm8 AVX512BW	
VPCMPW	Kreg | mask， zmmreg， zmmrm512， imm8 AVX512BW	
VPCOMPRESSD	mem128|mask,xmmreg	AVX512VL
VPCOMPRESSD	Mem128 | mask， xmmreg	AVX512VL
VPCOMPRESSD	mem256|mask,ymmreg	AVX512VL
VPCOMPRESSD	Mem256 | 面具， ymmreg	AVX512VL
VPCOMPRESSD	mem512|mask,zmmreg	AVX512
VPCOMPRESSD	Mem512 | mask， zmmreg	AVX512
VPCOMPRESSD	xmmreg|mask|z,xmmreg	AVX512VL
VPCOMPRESSD	| mask | z， xmmreg	AVX512VL
VPCOMPRESSD	ymmreg|mask|z,ymmreg	AVX512VL
VPCOMPRESSD	Ymmreg | mask | z， ymmreg	AVX512VL
VPCOMPRESSD	zmmreg|mask|z,zmmreg	AVX512
VPCOMPRESSD	Zmmreg | mask | z， zmmreg	AVX512
VPCOMPRESSQ	mem128|mask,xmmreg	AVX512VL
VPCOMPRESSQ	Mem128 | mask， xmmreg	AVX512VL
VPCOMPRESSQ	mem256|mask,ymmreg	AVX512VL
VPCOMPRESSQ	Mem256 | 面具， ymmreg	AVX512VL
VPCOMPRESSQ	mem512|mask,zmmreg	AVX512
VPCOMPRESSQ	Mem512 | mask， zmmreg	AVX512
VPCOMPRESSQ	xmmreg|mask|z,xmmreg	AVX512VL

VPCOMPRESSQ	| mask | z， xmmreg	AVX512VL
VPCOMPRESSQ	ymmreg|mask|z,ymmreg	AVX512VL
VPCOMPRESSQ	Ymmreg | mask | z， ymmreg	AVX512VL
VPCOMPRESSQ	zmmreg|mask|z,zmmreg	AVX512
VPCOMPRESSQ	Zmmreg | mask | z， zmmreg	AVX512
VPCONFLICTD	xmmreg|mask|z,xmmrm128|b32 AVX512VL/CD	
VPCONFLICTD	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL/CD	
VPCONFLICTD	ymmreg|mask|z,ymmrm256|b32 AVX512VL/CD	
VPCONFLICTD	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL/CD	
VPCONFLICTD	zmmreg|mask|z,zmmrm512|b32 AVX512CD	
VPCONFLICTD	Zmmrreg | mask | z， zmmrm512 | b32 AVX512CD	
VPCONFLICTQ	xmmreg|mask|z,xmmrm128|b64 AVX512VL/CD	
VPCONFLICTQ	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL/CD	
VPCONFLICTQ	ymmreg|mask|z,ymmrm256|b64 AVX512VL/CD	
VPCONFLICTQ	Ymmreg | mask | z， ymmrm256 | b64 AVX512VL/CD	
VPCONFLICTQ	zmmreg|mask|z,zmmrm512|b64 AVX512CD	
VPCONFLICTQ	Zmmrreg | mask | z， zmmrm512 | b64 AVX512CD	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/VBMI	
VPERMB	Xmmreg | mask | z， xmmreg * ， xmmrm128	
VPERMB	AVX512VL/VBMI	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/VBMI	
VPERMB	Ymmreg | mask | z， ymmreg * ， ymmrm256	
VPERMB	AVX512VL/VBMI	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512VBMI	
VPERMB	Zmmreg | mask | z， zmmreg * ， zmmrm512	
VPERMB	AVX512VBMI	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL	
VPERMD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32	
VPERMD	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512	
VPERMD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32	
VPERMD	AVX512	
	xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/VBMI	
VPERMI2B	Xmmreg | mask | z， xmmreg， xmmrm128	
VPERMI2B	AVX512VL/VBMI	
	ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/VBMI	
VPERMI2B	Ymmreg | mask | z， ymmreg， ymmrm256	
VPERMI2B	AVX512VL/VBMI	
VPERMI2B	zmmreg|mask|z,zmmreg,zmmrm512 AVX512VBMI	
VPERMI2B	Zmmreg | mask | z， zmmreg， zmmrm512 AVX512VBMI	
	xmmreg|mask|z,xmmreg,xmmrm128|b32	
VPERMI2D	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPERMI2D	| b32	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b32	
VPERMI2D	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPERMI2D	| b32	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b32	
VPERMI2D	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPERMI2D	| b32	AVX512
VPERMI2PD		AVX512VL
VPERMI2PD	xmmreg|mask|z,xmmreg,xmmrm128|b64	AVX512VL

Xmmreg | mask | z，xmmreg，xmmrm128
| b64
ymmreg|mask|z,ymmreg,ymmrm256|b64
VPERMI2PD Ymmreg | mask | z，ymmreg，ymmrm256 AVX512VL
VPERMI2PD | b64 AVX512VL
zmmreg|mask|z,zmmreg,zmmrm512|b64
VPERMI2PD Zmmreg | mask | z，zmmreg，zmmrm512 AVX512
VPERMI2PD | b64 AVX512
xmmreg|mask|z,xmmreg,xmmrm128|b32
VPERMI2PS Xmmreg | mask | z，xmmreg，xmmrm128 AVX512VL
VPERMI2PS | b32 AVX512VL
ymmreg|mask|z,ymmreg,ymmrm256|b32
VPERMI2PS Ymmreg | mask | z，ymmreg，ymmrm256 AVX512VL
VPERMI2PS | b32 AVX512VL
zmmreg|mask|z,zmmreg,zmmrm512|b32
VPERMI2PS Zmmreg | mask | z，zmmreg，zmmrm512 AVX512
VPERMI2PS | b32 AVX512
xmmreg|mask|z,xmmreg,xmmrm128|b64
VPERMI2Q Xmmreg | mask | z，xmmreg，xmmrm128 AVX512VL
VPERMI2Q | b64 AVX512VL
ymmreg|mask|z,ymmreg,ymmrm256|b64
VPERMI2Q Ymmreg | mask | z，ymmreg，ymmrm256 AVX512VL
VPERMI2Q | b64 AVX512VL
zmmreg|mask|z,zmmreg,zmmrm512|b64
VPERMI2Q Zmmreg | mask | z，zmmreg，zmmrm512 AVX512
VPERMI2Q | b64 AVX512
xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/BW
VPERMI2W Xmmreg | mask | z，xmmreg，xmmrm128
VPERMI2W AVX512VL/BW
ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/BW
VPERMI2W Ymmreg | mask | z，ymmreg，ymmrm256
VPERMI2W AVX512VL/BW
VPERMI2W zmmreg|mask|z,zmmreg,zmmrm512 AVX512BW
VPERMI2W Zmmreg | mask | z，zmmreg，zmmrm512 AVX512BW
VPERMILPD xmmreg|mask|z,xmmrm128|b64,imm8 AVX512VL
VPERMILPD Xmmreg | mask | z，xmmrm128 | b64，imm8 AVX512VL
VPERMILPD ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL
VPERMILPD Ymmreg | mask | z，ymmrm256 | b64，imm8 AVX512VL

216

VPERMILPD	zmmreg|mask|z,zmmrm512|b64,imm8 AVX512
VPERMILPD	Zmmrreg | mask | z， zmmrm512 | b64， imm8 AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPERMILPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPERMILPD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPERMILPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPERMILPD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPERMILPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPERMILPD	AVX512
	xmmreg|mask|z,xmmrm128|b32,imm8 AVX512VL
VPERMILPS	Xmmreg | mask | z， xmmrm128 | b32， imm8
VPERMILPS	AVX512VL
	ymmreg|mask|z,ymmrm256|b32,imm8 AVX512VL
VPERMILPS	Ymmreg | mask | z， ymmrm256 | b32， imm8
VPERMILPS	AVX512VL
VPERMILPS	zmmreg|mask|z,zmmrm512|b32,imm8 AVX512
VPERMILPS	Zmmrreg | mask | z， zmmrm512 | b32， imm8 AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPERMILPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPERMILPS	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPERMILPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPERMILPS	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPERMILPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPERMILPS	AVX512
	ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL
VPERMPD	Ymmreg | mask | z， ymmrm256 | b64， imm8
VPERMPD	AVX512VL
VPERMPD	zmmreg|mask|z,zmmrm512|b64,imm8 AVX512
VPERMPD	Zmmrreg | mask | z， zmmrm512 | b64， imm8 AVX512
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPERMPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPERMPD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPERMPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPERMPD	AVX512
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPERMPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPERMPS	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPERMPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPERMPS	AVX512
	ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL
VPERMQ	Ymmreg | mask | z， ymmrm256 | b64， imm8
VPERMQ	AVX512VL
VPERMQ	zmmreg|mask|z,zmmrm512|b64,imm8 AVX512
VPERMQ	Zmmrreg | mask | z， zmmrm512 | b64， imm8 AVX512

	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VPERMQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64	
VPERMQ	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512	
VPERMQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64	
VPERMQ	AVX512	
	xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/VBMI	
VPERMT2B	Xmmreg | mask | z， xmmreg， xmmrm128	
VPERMT2B	AVX512VL/VBMI	
	ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/VBMI	
VPERMT2B	Ymmreg | mask | z， ymmreg， ymmrm256	
VPERMT2B	AVX512VL/VBMI	
VPERMT2B	zmmreg|mask|z,zmmreg,zmmrm512 AVX512VBMI	
VPERMT2B	Zmmreg | mask | z， zmmreg， zmmrm512 AVX512VBMI	
	xmmreg|mask|z,xmmreg,xmmrm128|b32	
VPERMT2D	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPERMT2D	| b32	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b32	
VPERMT2D	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPERMT2D	| b32	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b32	
VPERMT2D	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPERMT2D	| b32	AVX512
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VPERMT2PD	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPERMT2PD	| b64	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VPERMT2PD	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPERMT2PD	| b64	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VPERMT2PD	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPERMT2PD	| b64	AVX512
VPERMT2PS	xmmreg|mask|z,xmmreg,xmmrm128|b32	
VPERMT2PS	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPERMT2PS	| b32	AVX512VL
VPERMT2PS	ymmreg|mask|z,ymmreg,ymmrm256|b32	
VPERMT2PS	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPERMT2PS	| b32	AVX512VL
VPERMT2PS	zmmreg|mask|z,zmmreg,zmmrm512|b32	
VPERMT2PS	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPERMT2PS	| b32	AVX512
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VPERMT2Q	Xmmreg | mask | z， xmmreg， xmmrm128	AVX512VL
VPERMT2Q	| b64	AVX512VL
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VPERMT2Q	Ymmreg | mask | z， ymmreg， ymmrm256	AVX512VL
VPERMT2Q	| b64	AVX512VL
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VPERMT2Q	Zmmreg | mask | z， zmmreg， zmmrm512	AVX512
VPERMT2Q	| b64	AVX512
VPERMT2W	xmmreg|mask|z,xmmreg,xmmrm128 AVX512VL/BW	

VPERMT2W	Xmmreg | mask | z， xmmreg， xmmrm128	
	AVX512VL/BW	
	ymmreg|mask|z,ymmreg,ymmrm256 AVX512VL/BW	
VPERMT2W	Ymmreg | mask | z， ymmreg， ymmrm256	
VPERMT2W	AVX512VL/BW	
VPERMT2W	zmmreg|mask|z,zmmreg,zmmrm512 AVX512BW	
VPERMT2W	Zmmreg | mask | z， zmmreg， zmmrm512 AVX512BW	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPERMW	Xmmreg | mask | z， xmmreg * ， xmmrm128	
VPERMW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPERMW	Ymmreg | mask | z， ymmreg * ， ymmrm256	
VPERMW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPERMW	Zmmreg | mask | z， zmmreg * ， zmmrm512	
VPERMW	AVX512BW	
	xmmreg|mask|z,mem128	
VPEXPANDD	Xmmreg | mask | z，	AVX512VL
Vpexpand	mem128	AVX512VL
	ymmreg|mask|z,mem256	
VPEXPANDD	Ymmreg | mask | z，	AVX512VL
Vpexpand	mem256	AVX512VL
	zmmreg|mask|z,mem512	
VPEXPANDD	Zmmreg | mask | z，	AVX512
Vpexpand	mem512	AVX512
VPEXPANDD	xmmreg|mask|z,xmmreg	AVX512VL
Vpexpand	| mask | z， xmmreg	AVX512VL
VPEXPANDD	ymmreg|mask|z,ymmreg	AVX512VL
Vpexpand	Ymmreg | mask | z， ymmreg	AVX512VL
VPEXPANDD	zmmreg|mask|z,zmmreg	AVX512
Vpexpand	Zmmreg | mask | z， zmmreg	AVX512
	xmmreg|mask|z,mem128	
VPEXPANDQ	Xmmreg | mask | z，	AVX512VL
VPEXPANDQ	mem128	AVX512VL
	ymmreg|mask|z,mem256	
VPEXPANDQ	Ymmreg | mask | z，	AVX512VL
VPEXPANDQ	mem256	AVX512VL
	zmmreg|mask|z,mem512	
VPEXPANDQ	Zmmreg | mask | z，	AVX512
VPEXPANDQ	mem512	AVX512
VPEXPANDQ	xmmreg|mask|z,xmmreg	AVX512VL
VPEXPANDQ	| mask | z， xmmreg	AVX512VL
VPEXPANDQ	ymmreg|mask|z,ymmreg	AVX512VL
VPEXPANDQ	Ymmreg | mask | z， ymmreg	AVX512VL
VPEXPANDQ	zmmreg|mask|z,zmmreg	AVX512
VPEXPANDQ	Zmmreg | mask | z， zmmreg	AVX512
VPEXTRB	reg8,xmmreg,imm8	AVX512BW
VPEXTRB	Reg8， xmmreg， imm8	AVX512BW

217

VPEXTRB	reg16,xmmreg,imm8	AVX512BW
VPEXTRB	Reg16， xmmreg， imm8	AVX512BW
VPEXTRB	reg32,xmmreg,imm8	AVX512BW
VPEXTRB	Reg32， xmmreg， imm8	AVX512BW
VPEXTRB	reg64,xmmreg,imm8	AVX512BW
VPEXTRB	Reg64， xmmreg， imm8	AVX512BW
VPEXTRB	mem8,xmmreg,imm8	AVX512BW
VPEXTRB	Mem8， xmmreg， imm8	AVX512BW
VPEXTRD	rm32,xmmreg,imm8	AVX512DQ
VPEXTRD	Rm32， xmmreg， imm8	AVX512DQ
VPEXTRQ	rm64,xmmreg,imm8	AVX512DQ
VPEXTRQ	Rm64， xmmreg， imm8	AVX512DQ
VPEXTRW	reg16,xmmreg,imm8	AVX512BW
VPEXTRW	Reg16， xmmreg， imm8	AVX512BW
VPEXTRW	reg32,xmmreg,imm8	AVX512BW
VPEXTRW	Reg32， xmmreg， imm8	AVX512BW
VPEXTRW	reg64,xmmreg,imm8	AVX512BW
VPEXTRW	Reg64， xmmreg， imm8	AVX512BW
VPEXTRW	mem16,xmmreg,imm8	AVX512BW
VPEXTRW	Mem16， xmmreg， imm8	AVX512BW
VPEXTRW	reg16,xmmreg,imm8	AVX512BW
VPEXTRW	Reg16， xmmreg， imm8	AVX512BW
VPEXTRW	reg32,xmmreg,imm8	AVX512BW
VPEXTRW	Reg32， xmmreg， imm8	AVX512BW
VPEXTRW	reg64,xmmreg,imm8	AVX512BW
VPEXTRW	Reg64， xmmreg， imm8	AVX512BW
VPGATHERDD	xmmreg|mask,xmem32	AVX512VL
Vpgather dd	Xmmreg | mask， xmem32	AVX512VL
VPGATHERDD	ymmreg|mask,ymem32	AVX512VL
Vpgather dd	Ymmreg | mask， ymem32	AVX512VL
VPGATHERDD	zmmreg|mask,zmem32	AVX512
Vpgather dd	Zmmreg | mask， zmem32	AVX512
VPGATHERDQ	xmmreg|mask,xmem64	AVX512VL
Vpgather dq	Xmmreg | mask， xmem64	AVX512VL
VPGATHERDQ	ymmreg|mask,xmem64	AVX512VL
Vpgather dq	Ymmreg | mask， xmem64	AVX512VL
VPGATHERDQ	zmmreg|mask,ymem64	AVX512
Vpgather dq	Zmmreg | mask， ymem64	AVX512
VPGATHERQD	xmmreg|mask,xmem32	AVX512VL
Vpgather qd	Xmmreg | mask， xmem32	AVX512VL
VPGATHERQD	xmmreg|mask,ymem32	AVX512VL
Vpgather qd	Xmmreg | mask， ymem32	AVX512VL
VPGATHERQD	ymmreg|mask,zmem32	AVX512
Vpgather qd	Ymmreg | mask， zmem32	AVX512
VPGATHERQQ	xmmreg|mask,xmem64	AVX512VL
Vpgather qq	Xmmreg | mask， xmem64	AVX512VL
VPGATHERQQ	ymmreg|mask,ymem64	AVX512VL
Vpgather qq	Ymmreg | mask， ymem64	AVX512VL
VPGATHERQQ	zmmreg|mask,zmem64	AVX512
Vpgather qq	Zmemreg | mask， zmem64	AVX512

	xmmreg,xmmreg*,reg32,imm8	AVX512BW
VPINSRB	Xmmreg，xmmreg *， reg32，imm8	
VPINSRB		AVX512BW
	xmmreg,xmmreg*,mem8,imm8	
VPINSRB	Xmmreg，xmmreg *， mem8，	AVX512BW
VPINSRB	imm8	AVX512BW
	xmmreg,xmmreg*,rm32,imm8	
VPINSRD	Xmmreg，xmmreg *， rm32，	AVX512DQ
VPINSRD	imm8	AVX512DQ
	xmmreg,xmmreg*,rm64,imm8	
VPINSRQ	Xmmreg，xmmreg *， rm64，	AVX512DQ
VPINSRQ	imm8	AVX512DQ
	xmmreg,xmmreg*,reg32,imm8	AVX512BW
VPINSRW	Xmmreg，xmmreg *， reg32，imm8	
VPINSRW		AVX512BW
	xmmreg,xmmreg*,mem16,imm8	AVX512BW
VPINSRW	Xmmreg，xmmreg *， mem16，imm8	
VPINSRW		AVX512BW
VPLZCNTD	xmmreg|mask|z,xmmrm128|b32	AVX512VL/CD
VPLZCNTD	Xmmreg | mask | z，xmmrm128 | b32	AVX512VL/CD
VPLZCNTD	ymmreg|mask|z,ymmrm256|b32	AVX512VL/CD
VPLZCNTD	Ymmreg | mask | z，ymmrm256 | b32	AVX512VL/CD
VPLZCNTD	zmmreg|mask|z,zmmrm512|b32	AVX512CD
VPLZCNTD	Zmmrreg | mask | z，zmmrm512 | b32	AVX512CD
VPLZCNTQ	xmmreg|mask|z,xmmrm128|b64	AVX512VL/CD
VPLZCNTQ	Xmmreg | mask | z，xmmrm128 | b64	AVX512VL/CD
VPLZCNTQ	ymmreg|mask|z,ymmrm256|b64	AVX512VL/CD
VPLZCNTQ	Ymmreg | mask | z，ymmrm256 | b64	AVX512VL/CD
VPLZCNTQ	zmmreg|mask|z,zmmrm512|b64	AVX512CD
VPLZCNTQ	Zmmrreg | mask | z，zmmrm512 | b64	AVX512CD
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VPMADD52HUQ	Xmmreg | mask | z，xmmreg，xmmrm128 |	AVX512VL/IFMA
VPMADD52HUQ	b64	AVX512VL/IFMA
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VPMADD52HUQ	Ymmreg | mask | z，ymmreg，ymmrm256 |	AVX512VL/IFMA
VPMADD52HUQ	b64	AVX512VL/IFMA
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VPMADD52HUQ	Zmmreg | mask | z，zmmreg，zmmrm512 |	AVX512IFMA
VPMADD52HUQ	b64	AVX512IFMA
	xmmreg|mask|z,xmmreg,xmmrm128|b64	
VPMADD52LUQ	Xmmreg | mask | z，xmmreg，xmmrm128 |	AVX512VL/IFMA
VPMADD52LUQ	b64	AVX512VL/IFMA
	ymmreg|mask|z,ymmreg,ymmrm256|b64	
VPMADD52LUQ	Ymmreg | mask | z，ymmreg，ymmrm256 |	AVX512VL/IFMA
VPMADD52LUQ	b64	AVX512VL/IFMA
	zmmreg|mask|z,zmmreg,zmmrm512|b64	
VPMADD52LUQ	Zmmreg | mask | z，zmmreg，zmmrm512 |	AVX512IFMA
VPMADD52LUQ	b64	AVX512IFMA
VPMADDUBSW	xmmreg|mask|z,xmmreg*,xmmrm128	AVX512VL/BW
VPMADDUBSW	Xmmreg | mask | z，xmmreg *， xmmrm128	AVX512VL/BW
VPMADDUBSW	ymmreg|mask|z,ymmreg*,ymmrm256	AVX512VL/BW

VPMADDUBSW	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPMADDUBSW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMADDUBSW	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPMADDWD	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMADDWD	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPMADDWD	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMADDWD	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPMADDWD	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMADDWD	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPMAXSB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMAXSB	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL/BW
VPMAXSB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMAXSB	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL/BW
VPMAXSB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMAXSB	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512BW
VPMAXSD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPMAXSD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPMAXSD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPMAXSD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL

218

	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPMAXSD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPMAXSD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPMAXSQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPMAXSQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPMAXSQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPMAXSQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPMAXSQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPMAXSQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMAXSW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPMAXSW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMAXSW	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMAXSW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMAXSW	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPMAXSW	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMAXUB	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPMAXUB	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMAXUB	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMAXUB	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMAXUB	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPMAXUB	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPMAXUD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPMAXUD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPMAXUD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPMAXUD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPMAXUD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPMAXUD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPMAXUQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPMAXUQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPMAXUQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPMAXUQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPMAXUQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPMAXUQ	AVX512
VPMAXUW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMAXUW	Xmmreg | mask | z， xmmreg * ， xmmrm128

	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMAXUW	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMAXUW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMAXUW	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPMAXUW	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMINSB	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPMINSB	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMINSB	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMINSB	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMINSB	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPMINSB	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPMINSD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPMINSD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPMINSD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPMINSD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPMINSD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPMINSD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPMINSQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPMINSQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPMINSQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPMINSQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPMINSQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPMINSQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMINSW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPMINSW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMINSW	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMINSW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMINSW	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPMINSW	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMINUB	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPMINUB	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMINUB	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPMINUB	AVX512VL/BW
VPMINUB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMINUB	Zmmreg | mask | z， zmmreg * ， zmmrm512

	AVX512BW	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL	
VPMINUD	Xmmreg | mask | z，xmmreg *，xmmrm128 | b32	
VPMINUD	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL	
VPMINUD	Ymmreg | mask | z，ymmreg *，ymmrm256 | b32	
VPMINUD	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512	
VPMINUD	Zmmreg | mask | z，zmmreg *，zmmrm512 | b32	
VPMINUD	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VPMINUQ	Xmmreg | mask | z，xmmreg *，xmmrm128 | b64	
VPMINUQ	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VPMINUQ	Ymmreg | mask | z，ymmreg *，ymmrm256 | b64	
VPMINUQ	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512	
VPMINUQ	Zmmreg | mask | z，zmmreg *，zmmrm512 | b64	
VPMINUQ	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPMINUW	Xmmreg | mask | z，xmmreg *，xmmrm128	
VPMINUW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPMINUW	Ymmreg | mask | z，ymmreg *，ymmrm256	
VPMINUW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPMINUW	Zmmreg | mask | z，zmmreg *，zmmrm512	
VPMINUW	AVX512BW	
VPMOVB2M	kreg,xmmreg	AVX512VL/BW
VPMOVB2M	Kreg，xmmreg	AVX512VL/BW
VPMOVB2M	kreg,ymmreg	AVX512VL/BW
VPMOVB2M	克雷格，伊姆雷格	AVX512VL/BW
VPMOVB2M	kreg,zmmreg	AVX512BW
VPMOVB2M	克雷格，zmmreg	AVX512BW
VPMOVD2M	kreg,xmmreg	AVX512VL/DQ
VPMOVD2M	Kreg，xmmreg	AVX512VL/DQ
VPMOVD2M	kreg,ymmreg	AVX512VL/DQ
VPMOVD2M	克雷格，伊姆雷格	AVX512VL/DQ
VPMOVD2M	kreg,zmmreg	AVX512DQ
VPMOVD2M	克雷格，zmmreg	AVX512DQ
VPMOVDB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVDB	| mask | z，xmmreg	AVX512VL
VPMOVDB	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVDB	Xmreg | mask | z，ymmreg	AVX512VL
VPMOVDB	xmmreg|mask|z,zmmreg	AVX512
VPMOVDB	Xmmreg | mask | z，zmmreg	AVX512
VPMOVDB	mem32|mask,xmmreg	AVX512VL
VPMOVDB	Mem32 | mask，xmmreg	AVX512VL
VPMOVDB	mem64|mask,ymmreg	AVX512VL
VPMOVDB	Mem64 | mask，ymmreg	AVX512VL

219

VPMOVDB	mem128|mask,zmmreg	AVX512
VPMOVDB	Mem128 | 面具， zmmreg	AVX512
VPMOVDW	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVDW	| mask | z， xmmreg	AVX512VL
VPMOVDW	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVDW	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVDW	ymmreg|mask|z,zmmreg	AVX512
VPMOVDW	Ymmreg | mask | z， zmreg	AVX512
VPMOVDW	mem64|mask,xmmreg	AVX512VL
VPMOVDW	Mem64 | mask， xmmreg	AVX512VL
VPMOVDW	mem128|mask,ymmreg	AVX512VL
VPMOVDW	Mem128 | mask， ymmreg	AVX512VL
VPMOVDW	mem256|mask,zmmreg	AVX512
VPMOVDW	Mem256 | mask， zmmreg	AVX512
VPMOVM2B	xmmreg,kreg	AVX512VL/BW
VPMOVM2B	Xmmreg， kreg	AVX512VL/BW
VPMOVM2B	ymmreg,kreg	AVX512VL/BW
VPMOVM2B	Ymmreg， kreg	AVX512VL/BW
VPMOVM2B	zmmreg,kreg	AVX512BW
VPMOVM2B	Zmmreg， kreg	AVX512BW
VPMOVM2D	xmmreg,kreg	AVX512VL/DQ
VPMOVM2D	Xmmreg， kreg	AVX512VL/DQ
VPMOVM2D	ymmreg,kreg	AVX512VL/DQ
VPMOVM2D	Ymmreg， kreg	AVX512VL/DQ
VPMOVM2D	zmmreg,kreg	AVX512DQ
VPMOVM2D	Zmmreg， kreg	AVX512DQ
VPMOVM2Q	xmmreg,kreg	AVX512VL/DQ
VPMOVM2Q	Xmmreg， kreg	AVX512VL/DQ
VPMOVM2Q	ymmreg,kreg	AVX512VL/DQ
VPMOVM2Q	Ymmreg， kreg	AVX512VL/DQ
VPMOVM2Q	zmmreg,kreg	AVX512DQ
VPMOVM2Q	Zmmreg， kreg	AVX512DQ
VPMOVM2W	xmmreg,kreg	AVX512VL/BW
VPMOVM2W	Xmmreg， kreg	AVX512VL/BW
VPMOVM2W	ymmreg,kreg	AVX512VL/BW
VPMOVM2W	Ymmreg， kreg	AVX512VL/BW
VPMOVM2W	zmmreg,kreg	AVX512BW
VPMOVM2W	Zmmreg， kreg	AVX512BW
VPMOVQ2M	kreg,xmmreg	AVX512VL/DQ
VPMOVQ2M	Kreg， xmmreg	AVX512VL/DQ
VPMOVQ2M	kreg,ymmreg	AVX512VL/DQ
VPMOVQ2M	克雷格，伊姆雷格	AVX512VL/DQ
VPMOVQ2M	kreg,zmmreg	AVX512DQ
VPMOVQ2M	克雷格， zmmreg	AVX512DQ
VPMOVQB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVQB	| mask | z， xmmreg	AVX512VL
VPMOVQB	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVQB	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVQB	xmmreg|mask|z,zmmreg	AVX512
VPMOVQB	Xmmreg | mask | z， zmmreg	AVX512

VPMOVQB	mem16|mask,xmmreg	AVX512VL
VPMOVQB	Mem16 | mask， xmmreg	AVX512VL
VPMOVQB	mem32|mask,ymmreg	AVX512VL
VPMOVQB	Mem32 | mask， ymmreg	AVX512VL
VPMOVQB	mem64|mask,zmmreg	AVX512
VPMOVQB	Mem64 | 面具， zmmreg	AVX512
VPMOVQD	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVQD	| mask | z， xmmreg	AVX512VL
VPMOVQD	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVQD	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVQD	ymmreg|mask|z,zmmreg	AVX512
VPMOVQD	Ymmreg | mask | z， zmreg	AVX512
VPMOVQD	mem64|mask,xmmreg	AVX512VL
VPMOVQD	Mem64 | mask， xmmreg	AVX512VL
VPMOVQD	mem128|mask,ymmreg	AVX512VL
VPMOVQD	Mem128 | mask， ymmreg	AVX512VL
VPMOVQD	mem256|mask,zmmreg	AVX512
VPMOVQD	Mem256 | mask， zmmreg	AVX512
VPMOVQW	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVQW	| mask | z， xmmreg	AVX512VL
VPMOVQW	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVQW	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVQW	xmmreg|mask|z,zmmreg	AVX512
VPMOVQW	Xmmreg | mask | z， zmmreg	AVX512
VPMOVQW	mem32|mask,xmmreg	AVX512VL
VPMOVQW	Mem32 | mask， xmmreg	AVX512VL
VPMOVQW	mem64|mask,ymmreg	AVX512VL
VPMOVQW	Mem64 | mask， ymmreg	AVX512VL
VPMOVQW	mem128|mask,zmmreg	AVX512
VPMOVQW	Mem128 | 面具， zmmreg	AVX512
VPMOVSDB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVSDB	| mask | z， xmmreg	AVX512VL
VPMOVSDB	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVSDB	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVSDB	xmmreg|mask|z,zmmreg	AVX512
VPMOVSDB	Xmmreg | mask | z， zmmreg	AVX512
VPMOVSDB	mem32|mask,xmmreg	AVX512VL
VPMOVSDB	Mem32 | mask， xmmreg	AVX512VL
VPMOVSDB	mem64|mask,ymmreg	AVX512VL
VPMOVSDB	Mem64 | mask， ymmreg	AVX512VL
VPMOVSDB	mem128|mask,zmmreg	AVX512
VPMOVSDB	Mem128 | 面具， zmmreg	AVX512
VPMOVSDW	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVSDW	| mask | z， xmmreg	AVX512VL
VPMOVSDW	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVSDW	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVSDW	ymmreg|mask|z,zmmreg	AVX512
VPMOVSDW	Ymmreg | mask | z， zmreg	AVX512
VPMOVSDW	mem64|mask,xmmreg	AVX512VL
VPMOVSDW	Mem64 | mask， xmmreg	AVX512VL

VPMOVSDW	mem128|mask,ymmreg	AVX512VL
VPMOVSDW	Mem128 | mask，ymmreg	AVX512VL
VPMOVSDW	mem256|mask,zmmreg	AVX512
VPMOVSDW	Mem256 | mask，zmmreg	AVX512
VPMOVSQB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVSQB	| mask | z，xmmreg	AVX512VL
VPMOVSQB	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVSQB	Xmreg | mask | z，ymmreg	AVX512VL

220

VPMOVSQB	xmmreg|mask|z,zmmreg	AVX512
VPMOVSQB	Xmmreg | mask | z， zmmreg	AVX512
VPMOVSQB	mem16|mask,xmmreg	AVX512VL
VPMOVSQB	Mem16 | mask， xmmreg	AVX512VL
VPMOVSQB	mem32|mask,ymmreg	AVX512VL
VPMOVSQB	Mem32 | mask， ymmreg	AVX512VL
VPMOVSQB	mem64|mask,zmmreg	AVX512
VPMOVSQB	Mem64 | 面具， zmmreg	AVX512
VPMOVSQD	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVSQD	| mask | z， xmmreg	AVX512VL
VPMOVSQD	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVSQD	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVSQD	ymmreg|mask|z,zmmreg	AVX512
VPMOVSQD	Ymmreg | mask | z， zmreg	AVX512
VPMOVSQD	mem64|mask,xmmreg	AVX512VL
VPMOVSQD	Mem64 | mask， xmmreg	AVX512VL
VPMOVSQD	mem128|mask,ymmreg	AVX512VL
VPMOVSQD	Mem128 | mask， ymmreg	AVX512VL
VPMOVSQD	mem256|mask,zmmreg	AVX512
VPMOVSQD	Mem256 | mask， zmmreg	AVX512
VPMOVSQW	xmmreg|mask|z,xmmreg	AVX512VL
Vmovsqw	| mask | z， xmmreg	AVX512VL
VPMOVSQW	xmmreg|mask|z,ymmreg	AVX512VL
Vmovsqw	Xmreg | mask | z， ymmreg	AVX512VL
VPMOVSQW	xmmreg|mask|z,zmmreg	AVX512
Vmovsqw	Xmmreg | mask | z， zmmreg	AVX512
VPMOVSQW	mem32|mask,xmmreg	AVX512VL
Vmovsqw	Mem32 | mask， xmmreg	AVX512VL
VPMOVSQW	mem64|mask,ymmreg	AVX512VL
Vmovsqw	Mem64 | mask， ymmreg	AVX512VL
VPMOVSQW	mem128|mask,zmmreg	AVX512
Vmovsqw	Mem128 | 面具， zmmreg	AVX512
VPMOVSWB	xmmreg|mask|z,xmmreg	AVX512VL/BW
VPMOVSWB	| mask | z， xmmreg	AVX512VL/BW
VPMOVSWB	xmmreg|mask|z,ymmreg	AVX512VL/BW
VPMOVSWB	Xmreg | mask | z， ymmreg	AVX512VL/BW
VPMOVSWB	ymmreg|mask|z,zmmreg	AVX512BW
VPMOVSWB	Ymmreg | mask | z， zmreg	AVX512BW
VPMOVSWB	mem64|mask,xmmreg	AVX512VL/BW
VPMOVSWB	Mem64 | mask， xmmreg	AVX512VL/BW
VPMOVSWB	mem128|mask,ymmreg	AVX512VL/BW
VPMOVSWB	Mem128 | mask， ymmreg	AVX512VL/BW
VPMOVSWB	mem256|mask,zmmreg	AVX512BW
VPMOVSWB	Mem256 | mask， zmmreg	AVX512BW
	xmmreg|mask|z,xmmrm32	
VPMOVSXBD	Xmmreg | mask | z，	AVX512VL
VPMOVSXBD	xmmrm32	AVX512VL
	ymmreg|mask|z,xmmrm64	
VPMOVSXBD	Ymmreg | mask | z，	AVX512VL
VPMOVSXBD	xmmrm64	AVX512VL

	zmmreg|mask|z,xmmrm128	
VPMOVSXBD	Zmmrreg | mask | z，	AVX512
VPMOVSXBD	xmmrm128	AVX512
	xmmreg|mask|z,xmmrm16	
VPMOVSXBQ	Xmmrreg | mask | z，	AVX512VL
VPMOVSXBQ	xmmrm16	AVX512VL
	ymmreg|mask|z,xmmrm32	
VPMOVSXBQ	Ymmreg | mask | z，	AVX512VL
VPMOVSXBQ	xmmrm32	AVX512VL
	zmmreg|mask|z,xmmrm64	
VPMOVSXBQ	Zmmrreg | mask | z，	AVX512
VPMOVSXBQ	xmmrm64	AVX512
	xmmreg|mask|z,xmmrm64	
VPMOVSXBW	Xmmreg | mask | z，	AVX512VL/BW
VPMOVSXBW	xmmrm64	AVX512VL/BW
	ymmreg|mask|z,xmmrm128	
VPMOVSXBW	Ymmreg | mask | z，	AVX512VL/BW
VPMOVSXBW	xmmrm128	AVX512VL/BW
	zmmreg|mask|z,ymmrm256	
VPMOVSXBW	Zmmreg | mask | z，	AVX512BW
VPMOVSXBW	ymmrm256	AVX512BW
	xmmreg|mask|z,xmmrm64	
VPMOVSXDQ	Xmmreg | mask | z，	AVX512VL
VPMOVSXDQ	xmmrm64	AVX512VL
	ymmreg|mask|z,xmmrm128	
VPMOVSXDQ	Ymmreg | mask | z，	AVX512VL
VPMOVSXDQ	xmmrm128	AVX512VL
	zmmreg|mask|z,ymmrm256	
VPMOVSXDQ	Zmmreg | mask | z，	AVX512
VPMOVSXDQ	ymmrm256	AVX512
	xmmreg|mask|z,xmmrm64	
VPMOVSXWD	Xmmreg | mask | z，	AVX512VL
VPMOVSXWD	xmmrm64	AVX512VL
	ymmreg|mask|z,xmmrm128	
VPMOVSXWD	Ymmreg | mask | z，	AVX512VL
VPMOVSXWD	xmmrm128	AVX512VL
	zmmreg|mask|z,ymmrm256	
VPMOVSXWD	Zmmreg | mask | z，	AVX512
VPMOVSXWD	ymmrm256	AVX512
	xmmreg|mask|z,xmmrm32	
VPMOVSXWQ	Xmmrreg | mask | z，	AVX512VL
VPMOVSXWQ	xmmrm32	AVX512VL
	ymmreg|mask|z,xmmrm64	
VPMOVSXWQ	Ymmreg | mask | z，	AVX512VL
VPMOVSXWQ	xmmrm64	AVX512VL
	zmmreg|mask|z,xmmrm128	
VPMOVSXWQ	Zmmrreg | mask | z，	AVX512
VPMOVSXWQ	xmmrm128	AVX512
VPMOVUSDB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVUSDB	| mask | z， xmmreg	AVX512VL
VPMOVUSDB	xmmreg|mask|z,ymmreg	AVX512VL

VPMOVUSDB	Xmreg | mask | z，ymmreg	AVX512VL
VPMOVUSDB	xmmreg|mask|z,zmmreg	AVX512
VPMOVUSDB	Xmmreg | mask | z，zmmreg	AVX512
VPMOVUSDB	mem32|mask,xmmreg	AVX512VL
VPMOVUSDB	Mem32 | mask，xmmreg	AVX512VL
VPMOVUSDB	mem64|mask,ymmreg	AVX512VL
VPMOVUSDB	Mem64 | mask，ymmreg	AVX512VL
VPMOVUSDB	mem128|mask,zmmreg	AVX512
VPMOVUSDB	Mem128 | 面具，zmmreg	AVX512
VPMOVUSDW	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVUSDW	| mask | z，xmmreg	AVX512VL
VPMOVUSDW	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVUSDW	Xmreg | mask | z，ymmreg	AVX512VL
VPMOVUSDW	ymmreg|mask|z,zmmreg	AVX512
VPMOVUSDW	Ymmreg | mask | z，zmreg	AVX512
VPMOVUSDW	mem64|mask,xmmreg	AVX512VL
VPMOVUSDW	Mem64 | mask，xmmreg	AVX512VL
VPMOVUSDW	mem128|mask,ymmreg	AVX512VL
VPMOVUSDW	Mem128 | mask，ymmreg	AVX512VL
VPMOVUSDW	mem256|mask,zmmreg	AVX512
VPMOVUSDW	Mem256 | mask，zmmreg	AVX512
VPMOVUSQB	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVUSQB	| mask | z，xmmreg	AVX512VL
VPMOVUSQB	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVUSQB	Xmreg | mask | z，ymmreg	AVX512VL

221

VPMOVUSQB	xmmreg|mask|z,zmmreg	AVX512
VPMOVUSQB	Xmmreg | mask | z，zmmreg	AVX512
VPMOVUSQB	mem16|mask,xmmreg	AVX512VL
VPMOVUSQB	Mem16 | mask，xmmreg	AVX512VL
VPMOVUSQB	mem32|mask,ymmreg	AVX512VL
VPMOVUSQB	Mem32 | mask，ymmreg	AVX512VL
VPMOVUSQB	mem64|mask,zmmreg	AVX512
VPMOVUSQB	Mem64 | 面具，zmmreg	AVX512
VPMOVUSQD	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVUSQD	| mask | z，xmmreg	AVX512VL
VPMOVUSQD	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVUSQD	Xmreg | mask | z，ymmreg	AVX512VL
VPMOVUSQD	ymmreg|mask|z,zmmreg	AVX512
VPMOVUSQD	Ymmreg | mask | z，zmreg	AVX512
VPMOVUSQD	mem64|mask,xmmreg	AVX512VL
VPMOVUSQD	Mem64 | mask，xmmreg	AVX512VL
VPMOVUSQD	mem128|mask,ymmreg	AVX512VL
VPMOVUSQD	Mem128 | mask，ymmreg	AVX512VL
VPMOVUSQD	mem256|mask,zmmreg	AVX512
VPMOVUSQD	Mem256 | mask，zmmreg	AVX512
VPMOVUSQW	xmmreg|mask|z,xmmreg	AVX512VL
VPMOVUSQW	| mask | z，xmmreg	AVX512VL
VPMOVUSQW	xmmreg|mask|z,ymmreg	AVX512VL
VPMOVUSQW	Xmreg | mask | z，ymmreg	AVX512VL
VPMOVUSQW	xmmreg|mask|z,zmmreg	AVX512
VPMOVUSQW	Xmmreg | mask | z，zmmreg	AVX512
VPMOVUSQW	mem32|mask,xmmreg	AVX512VL
VPMOVUSQW	Mem32 | mask，xmmreg	AVX512VL
VPMOVUSQW	mem64|mask,ymmreg	AVX512VL
VPMOVUSQW	Mem64 | mask，ymmreg	AVX512VL
VPMOVUSQW	mem128|mask,zmmreg	AVX512
VPMOVUSQW	Mem128 | 面具，zmmreg	AVX512
VPMOVUSWB	xmmreg|mask|z,xmmreg	AVX512VL/BW
VPMOVUSWB	| mask | z，xmmreg	AVX512VL/BW
VPMOVUSWB	xmmreg|mask|z,ymmreg	AVX512VL/BW
VPMOVUSWB	Xmreg | mask | z，ymmreg	AVX512VL/BW
VPMOVUSWB	ymmreg|mask|z,zmmreg	AVX512BW
VPMOVUSWB	Ymmreg | mask | z，zmreg	AVX512BW
VPMOVUSWB	mem64|mask,xmmreg	AVX512VL/BW
VPMOVUSWB	Mem64 | mask，xmmreg	AVX512VL/BW
VPMOVUSWB	mem128|mask,ymmreg	AVX512VL/BW
VPMOVUSWB	Mem128 | mask，ymmreg	AVX512VL/BW
VPMOVUSWB	mem256|mask,zmmreg	AVX512BW
VPMOVUSWB	Mem256 | mask，zmmreg	AVX512BW
VPMOVW2M	kreg,xmmreg	AVX512VL/BW
VPMOVW2M	Kreg，xmmreg	AVX512VL/BW
VPMOVW2M	kreg,ymmreg	AVX512VL/BW
VPMOVW2M	克雷格，伊姆雷格	AVX512VL/BW
VPMOVW2M	kreg,zmmreg	AVX512BW
VPMOVW2M	克雷格，zmmreg	AVX512BW

VPMOVWB	xmmreg|mask|z,xmmreg	AVX512VL/BW
VPMOVWB	| mask | z，xmmreg	AVX512VL/BW
VPMOVWB	xmmreg|mask|z,ymmreg	AVX512VL/BW
VPMOVWB	Xmreg | mask | z，ymmreg	AVX512VL/BW
VPMOVWB	ymmreg|mask|z,zmmreg	AVX512BW
VPMOVWB	Ymmreg | mask | z，zmreg	AVX512BW
VPMOVWB	mem64|mask,xmmreg	AVX512VL/BW
VPMOVWB	Mem64 | mask，xmmreg	AVX512VL/BW
VPMOVWB	mem128|mask,ymmreg	AVX512VL/BW
VPMOVWB	Mem128 | mask，ymmreg	AVX512VL/BW
VPMOVWB	mem256|mask,zmmreg	AVX512BW
VPMOVWB	Mem256 | mask，zmmreg	AVX512BW
	xmmreg|mask|z,xmmrm32	
VPMOVZXBD	Xmmrreg | mask | z，	AVX512VL
VPMOVZXBD	xmmrm32	AVX512VL
	ymmreg|mask|z,xmmrm64	
VPMOVZXBD	Ymmreg | mask | z，	AVX512VL
VPMOVZXBD	xmmrm64	AVX512VL
	zmmreg|mask|z,xmmrm128	
VPMOVZXBD	Zmmrreg | mask | z，	AVX512
VPMOVZXBD	xmmrm128	AVX512
	xmmreg|mask|z,xmmrm16	
VPMOVZXBQ	Xmmrreg | mask | z，	AVX512VL
VPMOVZXBQ	xmmrm16	AVX512VL
	ymmreg|mask|z,xmmrm32	
VPMOVZXBQ	Ymmreg | mask | z，	AVX512VL
VPMOVZXBQ	xmmrm32	AVX512VL
	zmmreg|mask|z,xmmrm64	
VPMOVZXBQ	Zmmrreg | mask | z，	AVX512
VPMOVZXBQ	xmmrm64	AVX512
	xmmreg|mask|z,xmmrm64	
VPMOVZXBW	Xmmreg | mask | z，	AVX512VL/BW
VPMOVZXBW	xmmrm64	AVX512VL/BW
	ymmreg|mask|z,xmmrm128	
VPMOVZXBW	Ymmreg | mask | z，	AVX512VL/BW
VPMOVZXBW	xmmrm128	AVX512VL/BW
	zmmreg|mask|z,ymmrm256	
VPMOVZXBW	Zmmreg | mask | z，	AVX512BW
VPMOVZXBW	ymmrm256	AVX512BW
	xmmreg|mask|z,xmmrm64	
VPMOVZXDQ	Xmmreg | mask | z，	AVX512VL
VPMOVZXDQ	xmmrm64	AVX512VL
	ymmreg|mask|z,xmmrm128	
VPMOVZXDQ	Ymmreg | mask | z，	AVX512VL
VPMOVZXDQ	xmmrm128	AVX512VL
	zmmreg|mask|z,ymmrm256	
VPMOVZXDQ	Zmmreg | mask | z，	AVX512
VPMOVZXDQ	ymmrm256	AVX512
	xmmreg|mask|z,xmmrm64	
VPMOVZXWD	Xmmreg | mask | z，	AVX512VL
VPMOVZXWD	xmmrm64	AVX512VL

	ymmreg|mask|z,xmmrm128	
VPMOVZXWD	Ymmreg | mask | z，	AVX512VL
VPMOVZXWD	xmmrm128	AVX512VL
	zmmreg|mask|z,ymmrm256	
VPMOVZXWD	Zmmreg | mask | z，	AVX512
VPMOVZXWD	ymmrm256	AVX512
	xmmreg|mask|z,xmmrm32	
VPMOVZXWQ	Xmmrreg | mask | z，	AVX512VL
VPMOVZXWQ	xmmrm32	AVX512VL
	ymmreg|mask|z,xmmrm64	
VPMOVZXWQ	Ymmreg | mask | z，	AVX512VL
VPMOVZXWQ	xmmrm64	AVX512VL
	zmmreg|mask|z,xmmrm128	
VPMOVZXWQ	Zmmrreg | mask | z，	AVX512
VPMOVZXWQ	xmmrm128	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VPMULDQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64	
VPMULDQ	AVX512VL	
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VPMULDQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64	
VPMULDQ	AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512	
VPMULDQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64	
VPMULDQ	AVX512	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPMULHRSW	Xmmreg | mask | z， xmmreg * ， xmmrm128	
VPMULHRSW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPMULHRSW	Ymmreg | mask | z， ymmreg * ， ymmrm256	
VPMULHRSW	AVX512VL/BW	

222

VPMULHRSW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMULHRSW	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
VPMULHUW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMULHUW	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPMULHUW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMULHUW	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPMULHUW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMULHUW	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
VPMULHW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPMULHW	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPMULHW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPMULHW	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPMULHW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPMULHW	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
VPMULLD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPMULLD	Xmmreg | mask | z， xmmreg *， xmmrm128 | b32 AVX512VL
VPMULLD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPMULLD	Ymmreg | mask | z， ymmreg *， ymmrm256 | b32 AVX512VL
VPMULLD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPMULLD	Zmmreg | mask | z， zmmreg *， zmmrm512 | b32 AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/DQ
VPMULLQ	Xmmreg | mask | z， xmmreg *， xmmrm128 | b64
VPMULLQ	AVX512VL/DQ
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/DQ
VPMULLQ	Ymmreg | mask | z， ymmreg *， ymmrm256 | b64
VPMULLQ	AVX512VL/DQ
VPMULLQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512DQ
VPMULLQ	Zmmreg | mask | z， zmmreg *， zmmrm512 | b64 AVX512DQ
VPMULLW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
Vmpmullw	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL/BW
VPMULLW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
Vmpmullw	Ymmreg | mask | z， ymmreg *， ymmrm256 AVX512VL/BW
VPMULLW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
Vmpmullw	Zmmreg | mask | z， zmmreg *， zmmrm512 AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/VBMI
VPMULTISHIFTQB	Xmmreg | mask | z， xmmreg *， xmmrm128 | b64
VPMULTISHIFTQB	AVX512VL/VBMI
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/VBMI
VPMULTISHIFTQB	Ymmreg | mask | z， ymmreg *， ymmrm256 | b64
VPMULTISHIFTQB	AVX512VL/VBMI
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512VBMI
VPMULTISHIFTQB	Zmmreg | mask | z， zmmreg *， zmmrm512 | b64
VPMULTISHIFTQB	AVX512VBMI
VPMULUDQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPMULUDQ	Xmmreg | mask | z， xmmreg *， xmmrm128 | b64 AVX512VL
VPMULUDQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPMULUDQ	Ymmreg | mask | z， ymmreg *， ymmrm256 | b64 AVX512VL
VPMULUDQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPMULUDQ	Zmmreg | mask | z， zmmreg *， zmmrm512 | b64 AVX512
VPORD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL

VPORD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPORD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPORD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VPORD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPORD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VPORQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPORQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VPORQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPORQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
VPORQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPORQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VPROLD	xmmreg|mask|z,xmmrm128|b32*,imm8 AVX512VL
VPROLD	Xmmreg | mask | z， xmmrm128 | b32 * ， imm8 AVX512VL
VPROLD	ymmreg|mask|z,ymmrm256|b32*,imm8 AVX512VL
VPROLD	Ymmreg | mask | z， ymmrm256 | b32 * ， imm8 AVX512VL
VPROLD	zmmreg|mask|z,zmmrm512|b32*,imm8 AVX512
VPROLD	Zmmrreg | mask | z， zmmrm512 | b32 * ， imm8 AVX512
VPROLQ	xmmreg|mask|z,xmmrm128|b64*,imm8 AVX512VL
VPROLQ	Xmmreg | mask | z， xmmrm128 | b64 * ， imm8 AVX512VL
VPROLQ	ymmreg|mask|z,ymmrm256|b64*,imm8 AVX512VL
VPROLQ	Ymmreg | mask | z， ymmrm256 | b64 * ， imm8 AVX512VL
VPROLQ	zmmreg|mask|z,zmmrm512|b64*,imm8 AVX512
VPROLQ	Zmmrreg | mask | z， zmmrm512 | b64 * ， imm8 AVX512
VPROLVD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPROLVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPROLVD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPROLVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VPROLVD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPROLVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VPROLVQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPROLVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VPROLVQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPROLVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
VPROLVQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPROLVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VPRORD	xmmreg|mask|z,xmmrm128|b32*,imm8 AVX512VL
VPRORD	Xmmreg | mask | z， xmmrm128 | b32 * ， imm8 AVX512VL
VPRORD	ymmreg|mask|z,ymmrm256|b32*,imm8 AVX512VL
VPRORD	Ymmreg | mask | z， ymmrm256 | b32 * ， imm8 AVX512VL
VPRORD	zmmreg|mask|z,zmmrm512|b32*,imm8 AVX512
VPRORD	Zmmrreg | mask | z， zmmrm512 | b32 * ， imm8 AVX512
VPRORQ	xmmreg|mask|z,xmmrm128|b64*,imm8 AVX512VL
VPRORQ	Xmmreg | mask | z， xmmrm128 | b64 * ， imm8 AVX512VL
VPRORQ	ymmreg|mask|z,ymmrm256|b64*,imm8 AVX512VL
VPRORQ	Ymmreg | mask | z， ymmrm256 | b64 * ， imm8 AVX512VL
VPRORQ	zmmreg|mask|z,zmmrm512|b64*,imm8 AVX512
VPRORQ	Zmmrreg | mask | z， zmmrm512 | b64 * ， imm8 AVX512
VPRORVD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPRORVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VPRORVD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL

VPRORVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL	
VPRORVD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512	
VPRORVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512	
VPRORVQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VPRORVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL	
VPRORVQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VPRORVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL	
VPRORVQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512	
VPRORVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512	
	xmmreg,xmmreg*,xmmrm128	
VPSADBW	Xmmreg， xmmreg * ，	AVX512VL/BW
VPSADBW	xmmrm128	AVX512VL/BW
	ymmreg,ymmreg*,ymmrm256	
VPSADBW	Ymmreg， ymmreg * ，	AVX512VL/BW
VPSADBW	ymmrm256	AVX512VL/BW

223

	zmmreg,zmmreg*,zmmrm512	
VPSADBW	Zmmreg， zmmreg *，	AVX512BW
VPSADBW	zmmrm512	AVX512BW
VPSCATTERDD	xmem32|mask,xmmreg	AVX512VL
[咒语]	Xmem32 | mask， xmmreg	AVX512VL
VPSCATTERDD	ymem32|mask,ymmreg	AVX512VL
[咒语]	Ymem32 | mask， ymmreg	AVX512VL
VPSCATTERDD	zmem32|mask,zmmreg	AVX512
[咒语]	Zmem32 | 面具， zmmreg	AVX512
VPSCATTERDQ	xmem64|mask,xmmreg	AVX512VL
VPSCATTERDQ	Xmem64 | mask， xmmreg	AVX512VL
VPSCATTERDQ	xmem64|mask,ymmreg	AVX512VL
VPSCATTERDQ	Xmem64 | mask， ymmreg	AVX512VL
VPSCATTERDQ	ymem64|mask,zmmreg	AVX512
VPSCATTERDQ	Ymem64 | 面具， zmmreg	AVX512
VPSCATTERQD	xmem32|mask,xmmreg	AVX512VL
VPSCATTERQD	Xmem32 | mask， xmmreg	AVX512VL
VPSCATTERQD	ymem32|mask,xmmreg	AVX512VL
VPSCATTERQD	Ymem32 | mask， xmmreg	AVX512VL
VPSCATTERQD	zmem32|mask,ymmreg	AVX512
VPSCATTERQD	Zmem32 | 面具， ymmreg	AVX512
VPSCATTERQQ	xmem64|mask,xmmreg	AVX512VL
VPSCATTERQQ	Xmem64 | mask， xmmreg	AVX512VL
VPSCATTERQQ	ymem64|mask,ymmreg	AVX512VL
VPSCATTERQQ	Ymem64 | 面具， ymmreg	AVX512VL
VPSCATTERQQ	zmem64|mask,zmmreg	AVX512
VPSCATTERQQ	Zmem64 | mask， zmmreg	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPSHUFB	Xmmreg | mask | z， xmmreg *， xmmrm128	
VPSHUFB	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPSHUFB	Ymmreg | mask | z， ymmreg *， ymmrm256	
VPSHUFB	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPSHUFB	Zmmreg | mask | z， zmmreg *， zmmrm512	
VPSHUFB	AVX512BW	
	xmmreg|mask|z,xmmrm128|b32,imm8 AVX512VL	
VPSHUFD	Xmmreg | mask | z， xmmrm128 | b32， imm8	
VPSHUFD	AVX512VL	
	ymmreg|mask|z,ymmrm256|b32,imm8 AVX512VL	
VPSHUFD	Ymmreg | mask | z， ymmrm256 | b32， imm8	
VPSHUFD	AVX512VL	
VPSHUFD	zmmreg|mask|z,zmmrm512|b32,imm8 AVX512	
VPSHUFD	Zmmrreg | mask | z， zmmrm512 | b32， imm8 AVX512	
VPSHUFHW	xmmreg|mask|z,xmmrm128,imm8 AVX512VL/BW	
这是什么意思	Xmmreg | mask | z， xmmrm128， imm8 AVX512VL/BW	
VPSHUFHW	ymmreg|mask|z,ymmrm256,imm8 AVX512VL/BW	
这是什么意思	Ymmreg | mask | z， ymmrm256， imm8 AVX512VL/BW	
VPSHUFHW	zmmreg|mask|z,zmmrm512,imm8 AVX512BW	
这是什么意思	Zmmrreg | mask | z， zmmrm512， imm8 AVX512BW	

VPSHUFLW	xmmreg|mask|z,xmmrm128,imm8 AVX512VL/BW
(翻译)	Xmmreg | mask | z， xmmrm128， imm8 AVX512VL/BW
VPSHUFLW	ymmreg|mask|z,ymmrm256,imm8 AVX512VL/BW
(翻译)	Ymmreg | mask | z， ymmrm256， imm8 AVX512VL/BW
VPSHUFLW	zmmreg|mask|z,zmmrm512,imm8 AVX512BW
(翻译)	Zmmrreg | mask | z， zmmrm512， imm8 AVX512BW
VPSLLD	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL
VPSLLD	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VPSLLD	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL
VPSLLD	Ymmreg | mask | z， ymmreg * ， xmmrm128 AVX512VL
VPSLLD	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512
VPSLLD	Zmmreg | mask | z， zmmreg * ， xmmrm128 AVX512
	xmmreg|mask|z,xmmrm128|b32*,imm8 AVX512VL
VPSLLD	Xmmreg | mask | z， xmmrm128 | b32 * ， imm8
VPSLLD	AVX512VL
	ymmreg|mask|z,ymmrm256|b32*,imm8 AVX512VL
VPSLLD	Ymmreg | mask | z， ymmrm256 | b32 * ， imm8
VPSLLD	AVX512VL
	zmmreg|mask|z,zmmrm512|b32*,imm8 AVX512
VPSLLD	Zmmrreg | mask | z， zmmrm512 | b32 * ， imm8
VPSLLD	AVX512
	xmmreg,xmmrm128*,imm8
VPSLLDQ	Xmmreg， xmmrm128 * ， AVX512VL/BW
VPSLLDQ	imm8 AVX512VL/BW
	ymmreg,ymmrm256*,imm8
VPSLLDQ	Ymmreg， ymmrm256 * ， AVX512VL/BW
VPSLLDQ	imm8 AVX512VL/BW
	zmmreg,zmmrm512*,imm8
VPSLLDQ	Zmmrreg， zmmrm512 * ， AVX512BW
VPSLLDQ	imm8 AVX512BW
VPSLLQ	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL
VPSLLQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VPSLLQ	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL
VPSLLQ	Ymmreg | mask | z， ymmreg * ， xmmrm128 AVX512VL
VPSLLQ	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512
VPSLLQ	Zmmreg | mask | z， zmmreg * ， xmmrm128 AVX512
	xmmreg|mask|z,xmmrm128|b64*,imm8 AVX512VL
VPSLLQ	Xmmreg | mask | z， xmmrm128 | b64 * ， imm8
VPSLLQ	AVX512VL
	ymmreg|mask|z,ymmrm256|b64*,imm8 AVX512VL
VPSLLQ	Ymmreg | mask | z， ymmrm256 | b64 * ， imm8
VPSLLQ	AVX512VL
	zmmreg|mask|z,zmmrm512|b64*,imm8 AVX512
VPSLLQ	Zmmrreg | mask | z， zmmrm512 | b64 * ， imm8
VPSLLQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPSLLVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPSLLVD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPSLLVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPSLLVD	AVX512VL

	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPSLLVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPSLLVD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPSLLVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPSLLVQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPSLLVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPSLLVQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPSLLVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPSLLVQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSLLVW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPSLLVW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSLLVW	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPSLLVW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSLLVW	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPSLLVW	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSLLW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPSLLW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL/BW
VPSLLW	Ymmreg | mask | z， ymmreg * ， xmmrm128
VPSLLW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512BW
VPSLLW	Zmmreg | mask | z， zmmreg * ， xmmrm128
VPSLLW	AVX512BW
	xmmreg|mask|z,xmmrm128*,imm8 AVX512VL/BW
VPSLLW	Xmmreg | mask | z， xmmrm128 * ， imm8
VPSLLW	AVX512VL/BW
	ymmreg|mask|z,ymmrm256*,imm8 AVX512VL/BW
VPSLLW	Ymmreg | mask | z， ymmrm256 * ， imm8
VPSLLW	AVX512VL/BW

224

VPSLLW	zmmreg|mask|z,zmmrm512*,imm8 AVX512BW
VPSLLW	Zmmreg | mask | z， zmmrm512 * ， imm8 AVX512BW
VPSRAD	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL
VPSRAD	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VPSRAD	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL
VPSRAD	Ymmreg | mask | z， ymmreg * ， xmmrm128 AVX512VL
VPSRAD	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512
VPSRAD	Zmmreg | mask | z， zmmreg * ， xmmrm128 AVX512
	xmmreg|mask|z,xmmrm128|b32*,imm8 AVX512VL
VPSRAD	Xmmreg | mask | z， xmmrm128 | b32 * ， imm8
VPSRAD	AVX512VL
	ymmreg|mask|z,ymmrm256|b32*,imm8 AVX512VL
VPSRAD	Ymmreg | mask | z， ymmrm256 | b32 * ， imm8
VPSRAD	AVX512VL
	zmmreg|mask|z,zmmrm512|b32*,imm8 AVX512
VPSRAD	Zmmrreg | mask | z， zmmrm512 | b32 * ， imm8
VPSRAD	AVX512
VPSRAQ	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL
VPSRAQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VPSRAQ	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL
VPSRAQ	Ymmreg | mask | z， ymmreg * ， xmmrm128 AVX512VL
VPSRAQ	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512
VPSRAQ	Zmmreg | mask | z， zmmreg * ， xmmrm128 AVX512
	xmmreg|mask|z,xmmrm128|b64*,imm8 AVX512VL
VPSRAQ	Xmmreg | mask | z， xmmrm128 | b64 * ， imm8
VPSRAQ	AVX512VL
	ymmreg|mask|z,ymmrm256|b64*,imm8 AVX512VL
VPSRAQ	Ymmreg | mask | z， ymmrm256 | b64 * ， imm8
VPSRAQ	AVX512VL
	zmmreg|mask|z,zmmrm512|b64*,imm8 AVX512
VPSRAQ	Zmmrreg | mask | z， zmmrm512 | b64 * ， imm8
VPSRAQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPSRAVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPSRAVD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPSRAVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPSRAVD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPSRAVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPSRAVD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPSRAVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPSRAVQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPSRAVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPSRAVQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPSRAVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPSRAVQ	AVX512

	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPSRAVW	Xmmreg | mask | z， xmmreg *， xmmrm128	
VPSRAVW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW	
VPSRAVW	Ymmreg | mask | z， ymmreg *， ymmrm256	
VPSRAVW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW	
VPSRAVW	Zmmreg | mask | z， zmmreg *， zmmrm512	
VPSRAVW	AVX512BW	
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW	
VPSRAW	Xmmreg | mask | z， xmmreg *， xmmrm128	
VPSRAW	AVX512VL/BW	
	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL/BW	
VPSRAW	Ymmreg | mask | z， ymmreg *， xmmrm128	
VPSRAW	AVX512VL/BW	
	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512BW	
VPSRAW	Zmmreg | mask | z， zmmreg *， xmmrm128	
VPSRAW	AVX512BW	
	xmmreg|mask|z,xmmrm128*,imm8 AVX512VL/BW	
VPSRAW	Xmmreg | mask | z， xmmrm128 *， imm8	
VPSRAW	AVX512VL/BW	
	ymmreg|mask|z,ymmrm256*,imm8 AVX512VL/BW	
VPSRAW	Ymmreg | mask | z， ymmrm256 *， imm8	
VPSRAW	AVX512VL/BW	
VPSRAW	zmmreg|mask|z,zmmrm512*,imm8 AVX512BW	
VPSRAW	Zmmreg | mask | z， zmmrm512 *， imm8 AVX512BW	
VPSRLD	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL	
VPSRLD	Xmmreg | mask | z， xmmreg *， xmmrm128 AVX512VL	
VPSRLD	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL	
VPSRLD	Ymmreg | mask | z， ymmreg *， xmmrm128 AVX512VL	
VPSRLD	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512	
VPSRLD	Zmmreg | mask | z， zmmreg *， xmmrm128 AVX512	
	xmmreg|mask|z,xmmrm128|b32*,imm8 AVX512VL	
VPSRLD	Xmmreg | mask | z， xmmrm128 | b32 *， imm8	
VPSRLD	AVX512VL	
	ymmreg|mask|z,ymmrm256|b32*,imm8 AVX512VL	
VPSRLD	Ymmreg | mask | z， ymmrm256 | b32 *， imm8	
VPSRLD	AVX512VL	
	zmmreg|mask|z,zmmrm512|b32*,imm8 AVX512	
VPSRLD	Zmmrreg | mask | z， zmmrm512 | b32 *， imm8	
VPSRLD	AVX512	
	xmmreg,xmmrm128*,imm8	
VPSRLDQ	Xmmreg， xmmrm128 *，	AVX512VL/BW
VPSRLDQ	imm8	AVX512VL/BW
	ymmreg,ymmrm256*,imm8	
VPSRLDQ	Ymmreg， ymmrm256 *，	AVX512VL/BW
VPSRLDQ	imm8	AVX512VL/BW
	zmmreg,zmmrm512*,imm8	
VPSRLDQ	Zmmrreg， zmmrm512 *，	AVX512BW
VPSRLDQ	imm8	AVX512BW
VPSRLQ	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL	

VPSRLQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL
VPSRLQ	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL
VPSRLQ	Ymmreg | mask | z， ymmreg * ， xmmrm128 AVX512VL
VPSRLQ	zmmreg|mask|z,zmmreg*,xmmrm128 AVX512
VPSRLQ	Zmmreg | mask | z， zmmreg * ， xmmrm128 AVX512
	xmmreg|mask|z,xmmrm128|b64*,imm8 AVX512VL
VPSRLQ	Xmmreg | mask | z， xmmrm128 | b64 * ， imm8
VPSRLQ	AVX512VL
	ymmreg|mask|z,ymmrm256|b64*,imm8 AVX512VL
VPSRLQ	Ymmreg | mask | z， ymmrm256 | b64 * ， imm8
VPSRLQ	AVX512VL
	zmmreg|mask|z,zmmrm512|b64*,imm8 AVX512
VPSRLQ	Zmmrreg | mask | z， zmmrm512 | b64 * ， imm8
VPSRLQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPSRLVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VPSRLVD	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPSRLVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VPSRLVD	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPSRLVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32
VPSRLVD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPSRLVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VPSRLVQ	AVX512VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPSRLVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VPSRLVQ	AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPSRLVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64
VPSRLVQ	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSRLVW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPSRLVW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSRLVW	Ymmreg | mask | z， ymmreg * ， ymmrm256
VPSRLVW	AVX512VL/BW
	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSRLVW	Zmmreg | mask | z， zmmreg * ， zmmrm512
VPSRLVW	AVX512BW
	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSRLW	Xmmreg | mask | z， xmmreg * ， xmmrm128
VPSRLW	AVX512VL/BW
	ymmreg|mask|z,ymmreg*,xmmrm128 AVX512VL/BW
VPSRLW	Ymmreg | mask | z， ymmreg * ， xmmrm128
VPSRLW	AVX512VL/BW

225

```
VPSRLW           zmmreg|mask|z,zmmreg*,xmmrm128 AVX512BW
VPSRLW zmmreg | mask | z, zmmreg * , xmmrm128 AVX512BW
VPSRLW           xmmreg|mask|z,xmmrm128*,imm8 AVX512VL/BW
VPSRLW xmmreg | mask | z, xmmrm128 * , imm8 AVX512VL/BW
VPSRLW           ymmreg|mask|z,ymmrm256*,imm8 AVX512VL/BW
VPSRLW ymmreg | mask | z, ymmrm256 * , imm8 AVX512VL/BW
VPSRLW           zmmreg|mask|z,zmmrm512*,imm8 AVX512BW
VPSRLW zmmreg | mask | z, zmmrm512 * , imm8 AVX512BW
VPSUBB           xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBB xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBB           ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSUBB ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBB           zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBB zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPSUBD           xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPSUBD xmmreg | mask | z, xmmreg * , xmmrm128 | b32 AVX512VL
VPSUBD           ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPSUBD ymmreg | mask | z, ymmreg * , ymmrm256 | b32 AVX512VL
VPSUBD           zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPSUBD zmmreg | mask | z, zmmreg * , zmmrm512 | b32 AVX512
VPSUBQ           xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPSUBQ xmmreg | mask | z, xmmreg * , xmmrm128 | b64 AVX512VL
VPSUBQ           ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPSUBQ ymmreg | mask | z, ymmreg * , ymmrm256 | b64 AVX512VL
VPSUBQ           zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPSUBQ zmreg | mask | z, zmreg * , zmmrm512 | b64 AVX512
VPSUBSB          xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBSB xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBSB          ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSUBSB ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBSB          zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBSB zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPSUBSW          xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBSW xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBSW          ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSUBSW ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBSW          zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBSW zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPSUBUSB         xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBUSB xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBUSB         ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSUBUSB ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBUSB         zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBUSB zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPSUBUSW         xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBUSW xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBUSW         ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPSUBUSW ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBUSW         zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBUSW zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPSUBW           xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPSUBW xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPSUBW           ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
Ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPSUBW           zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPSUBW zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPTERNLOGD        xmmreg|mask|z,xmmreg,xmmrm128|b32,imm8 AVX512VL
VPTERNLOGD xmmreg | mask | z, xmmreg, xmmrm128 | b32, imm8 AVX512VL
VPTERNLOGD        ymmreg|mask|z,ymmreg,ymmrm256|b32,imm8 AVX512VL
VPTERNLOGD ymmreg | mask | z, ymmreg, ymmrm256 | b32, imm8 AVX512VL
VPTERNLOGD        zmmreg|mask|z,zmmreg,zmmrm512|b32,imm8 AVX512
VPTERNLOGD zmmreg | mask | z, zmmreg, zmmrm512 | b32, imm8 AVX512
VPTERNLOGQ        xmmreg|mask|z,xmmreg,xmmrm128|b64,imm8 AVX512VL
VPTERNLOGQ xmmreg | mask | z, xmmreg, xmmrm128 | b64, imm8 AVX512VL
VPTERNLOGQ        ymmreg|mask|z,ymmreg,ymmrm256|b64,imm8 AVX512VL
VPTERNLOGQ ymmreg | mask | z, ymmreg, ymmrm256 | b64, imm8 AVX512VL
VPTERNLOGQ        zmmreg|mask|z,zmmreg,zmmrm512|b64,imm8 AVX512
VPTERNLOGQ zmmreg | mask | z, zmmreg, zmmrm512 | b64, imm8 AVX512
VPTESTMB          kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPTESTMB kreg | mask, xmmreg, xmmrm128 AVX512VL/BW
VPTESTMB          kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPTESTMB kreg | mask, ymmreg, ymmrm256 AVX512VL/BW
VPTESTMB          kreg|mask,zmmreg,zmmrm512 AVX512BW
VPTESTMB kreg | mask, zmreg, zmmrm512 AVX512BW
VPTESTMD          kreg|mask,xmmreg,xmmrm128|b32 AVX512VL
VPTESTMD kreg | mask, xmmreg, xmmrm128 | b32 AVX512VL
VPTESTMD          kreg|mask,ymmreg,ymmrm256|b32 AVX512VL
VPTESTMD kreg | mask, ymmreg, ymmrm256 | b32 AVX512VL
VPTESTMD          kreg|mask,zmmreg,zmmrm512|b32 AVX512
VPTESTMD kreg | 面具, zmmreg, zmmrm512 | b32 AVX512
VPTESTMQ          kreg|mask,xmmreg,xmmrm128|b64 AVX512VL
VPTESTMQ kreg | mask, xmmreg, xmmrm128 | b64 AVX512VL
VPTESTMQ          kreg|mask,ymmreg,ymmrm256|b64 AVX512VL
VPTESTMQ kreg | mask, ymmreg, ymmrm256 | b64 AVX512VL
VPTESTMQ          kreg|mask,zmmreg,zmmrm512|b64 AVX512
VPTESTMQ kreg | mask, zmreg, zmmrm512 | b64 AVX512
VPTESTMW          kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPTESTMW kreg | mask, xmmreg, xmmrm128 AVX512VL/BW
VPTESTMW          kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPTESTMW kreg | mask, ymmreg, ymmrm256 AVX512VL/BW
VPTESTMW          kreg|mask,zmmreg,zmmrm512 AVX512BW
VPTESTMW kreg | mask, zmmreg, zmmrm512 AVX512BW
VPTESTNMB         kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPTESTNMB kreg | mask, xmmreg, xmmrm128 AVX512VL/BW
VPTESTNMB         kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPTESTNMB kreg | mask, ymmreg, ymmrm256 AVX512VL/BW
VPTESTNMB         kreg|mask,zmmreg,zmmrm512 AVX512BW
VPTESTNMB kreg | mask, zmreg, zmmrm512 AVX512BW
VPTESTNMD         kreg|mask,xmmreg,xmmrm128|b32 AVX512VL
VPTESTNMD kreg | mask, xmmreg, xmmrm128 | b32 AVX512VL
VPTESTNMD         kreg|mask,ymmreg,ymmrm256|b32 AVX512VL
VPTESTNMD kreg | mask, ymmreg, ymmrm256 | b32 AVX512VL
VPTESTNMD         kreg|mask,zmmreg,zmmrm512|b32 AVX512
VPTESTNMD kreg | 面具, zmmreg, zmmrm512 | b32 AVX512
VPTESTNMQ         kreg|mask,xmmreg,xmmrm128|b64 AVX512VL
VPTESTNMQ kreg | mask, xmmreg, xmmrm128 | b64 AVX512VL
VPTESTNMQ         kreg|mask,ymmreg,ymmrm256|b64 AVX512VL
VPTESTNMQ kreg | mask, ymmreg, ymmrm256 | b64 AVX512VL
```

226

```
VPTESTNMQ        kreg|mask,zmmreg,zmmrm512|b64 AVX512
VPTESTNMQ kreg | mask, zmreg, zmmrm512 | b64 AVX512
VPTESTNMW        kreg|mask,xmmreg,xmmrm128 AVX512VL/BW
VPTESTNMW kreg | mask, xmmreg, xmmrm128 AVX512VL/BW
VPTESTNMW        kreg|mask,ymmreg,ymmrm256 AVX512VL/BW
VPTESTNMW kreg | mask, ymmreg, ymmrm256 AVX512VL/BW
VPTESTNMW        kreg|mask,zmmreg,zmmrm512 AVX512BW
VPTESTNMW kreg | mask, zmmreg, zmmrm512 AVX512BW
VPUNPCKHBW       xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPUNPCKHBW xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPUNPCKHBW       ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPUNPCKHBW ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPUNPCKHBW       zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPUNPCKHBW zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPUNPCKHDQ       xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPUNPCKHDQ xmmreg | mask | z, xmmreg * , xmmrm128 | b32 AVX512VL
VPUNPCKHDQ       ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPUNPCKHDQ ymmreg | mask | z, ymmreg * , ymmrm256 | b32 AVX512VL
VPUNPCKHDQ       zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPUNPCKHDQ zmmreg | mask | z, zmmreg * , zmmrm512 | b32 AVX512
VPUNPCKHQDQ      xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPUNPCKHQDQ xmmreg | mask | z, xmmreg * , xmmrm128 | b64 AVX512VL
VPUNPCKHQDQ      ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPUNPCKHQDQ ymmreg | mask | z, ymmreg * , ymmrm256 | b64 AVX512VL
VPUNPCKHQDQ      zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPUNPCKHQDQ zmmreg | mask | z, zmmreg * , zmmrm512 | b64 AVX512
VPUNPCKHWD       xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPUNPCKHWD xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPUNPCKHWD       ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPUNPCKHWD ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPUNPCKHWD       zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPUNPCKHWD zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPUNPCKLBW       xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPUNPCKLBW xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPUNPCKLBW       ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPUNPCKLBW ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPUNPCKLBW       zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPUNPCKLBW zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPUNPCKLDQ       xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPUNPCKLDQ xmmreg | mask | z, xmmreg * , xmmrm128 | b32 AVX512VL
VPUNPCKLDQ       ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPUNPCKLDQ ymmreg | mask | z, ymmreg * , ymmrm256 | b32 AVX512VL
VPUNPCKLDQ       zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPUNPCKLDQ zmmreg | mask | z, zmmreg * , zmmrm512 | b32 AVX512
VPUNPCKLQDQ      xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPUNPCKLQDQ xmmreg | mask | z, xmmreg * , xmmrm128 | b64 AVX512VL
VPUNPCKLQDQ      ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPUNPCKLQDQ ymmreg | mask | z, ymmreg * , ymmrm256 | b64 AVX512VL
VPUNPCKLQDQ      zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPUNPCKLQDQ zmmreg | mask | z, zmmreg * , zmmrm512 | b64 AVX512
VPUNPCKLWD       xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL/BW
VPUNPCKLWD xmmreg | mask | z, xmmreg * , xmmrm128 AVX512VL/BW
VPUNPCKLWD       ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL/BW
VPUNPCKLWD ymmreg | mask | z, ymmreg * , ymmrm256 AVX512VL/BW
VPUNPCKLWD       zmmreg|mask|z,zmmreg*,zmmrm512 AVX512BW
VPUNPCKLWD zmmreg | mask | z, zmmreg * , zmmrm512 AVX512BW
VPXORD           xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VPXORD xmmreg | mask | z, xmmreg * , xmmrm128 | b32 AVX512VL
VPXORD           ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VPXORD ymmreg | mask | z, ymmreg * , ymmrm256 | b32 AVX512VL
VPXORD           zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VPXORD zmmreg | mask | z, zmmreg * , zmmrm512 | b32 AVX512
VPXORQ           xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VPXORQ xmmreg | mask | z, xmmreg * , xmmrm128 | b64 AVX512VL
VPXORQ           ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VPXORQ ymmreg | mask | z, ymmreg * , ymmrm256 | b64 AVX512VL
VPXORQ           zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VPXORQ zmmreg | mask | z, zmmreg * , zmmrm512 | b64 AVX512
VRANGEPD         xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VL/DQ
VRANGEPD xmmreg | mask | z, xmmreg * , xmmrm128 | b64, imm8 AVX512VL/DQ
VRANGEPD         ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL/DQ
VRANGEPD ymmreg | mask | z, ymmreg * , ymmrm256 | b64, imm8 AVX512VL/DQ
VRANGEPD         zmmreg|mask|z,zmmreg*,zmmrm512|b64|sae,imm8 AVX512DQ
VRANGEPD zmmreg | mask | z, zmmreg * , zmmrm512 | b64 | sae, imm8 AVX512DQ
VRANGEPS         xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8 AVX512VL/DQ
VRANGEPS xmmreg | mask | z, xmmreg * , xmmrm128 | b32, imm8 AVX512VL/DQ
VRANGEPS         ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL/DQ
VRANGEPS ymmreg | mask | z, ymmreg * , ymmrm256 | b32, imm8 AVX512VL/DQ
VRANGEPS         zmmreg|mask|z,zmmreg*,zmmrm512|b32|sae,imm8 AVX512DQ
VRANGEPS zmmreg | mask | z, zmmreg * , zmmrm512 | b32 | sae, imm8 AVX512DQ
VRANGESD         xmmreg|mask|z,xmmreg*,xmmrm64|sae,imm8 AVX512DQ
VRANGESD xmmreg | mask | z, xmmreg * , xmmrm64 | sae, imm8 AVX512DQ
VRANGESS         xmmreg|mask|z,xmmreg*,xmmrm32|sae,imm8 AVX512DQ
范围 xmmreg | mask | z, xmmreg * , xmmrm32 | sae, imm8 AVX512DQ
VRCP14PD         xmmreg|mask|z,xmmrm128|b64 AVX512VL
VRCP14PD xmmreg | mask | z, xmmrm128 | b64 AVX512VL
VRCP14PD         ymmreg|mask|z,ymmrm256|b64 AVX512VL
VRCP14PD ymmreg | mask | z, ymmrm256 | b64 AVX512VL
VRCP14PD         zmmreg|mask|z,zmmrm512|b64 AVX512
VRCP14PD zmreg | mask | z, zmmrm512 | b64 AVX512
VRCP14PS         xmmreg|mask|z,xmmrm128|b32 AVX512VL
VRCP14PS xmmreg | mask | z, xmmrm128 | b32 AVX512VL
VRCP14PS         ymmreg|mask|z,ymmrm256|b32 AVX512VL
VRCP14PS ymmreg | mask | z, ymmrm256 | b32 AVX512VL
VRCP14PS         zmmreg|mask|z,zmmrm512|b32 AVX512
VRCP14PS zmreg | mask | z, zmmrm512 | b32 AVX512
VRCP14SD         xmmreg|mask|z,xmmreg*,xmmrm64 AVX512
VRCP14SD xmmreg | mask | z, xmmreg * , xmmrm64 AVX512
VRCP14SS         xmmreg|mask|z,xmmreg*,xmmrm32 AVX512
VRCP14SS xmmreg | mask | z, xmmreg * , xmmrm32 AVX512
VRCP28PD         zmmreg|mask|z,zmmrm512|b64|sae AVX512ER
VRCP28PD zmmreg | mask | z, zmmrm512 | b64 | sae AVX512ER
VRCP28PS         zmmreg|mask|z,zmmrm512|b32|sae AVX512ER
VRCP28PS zmmreg | mask | z, zmmrm512 | b32 | sae AVX512ER
VRCP28SD         xmmreg|mask|z,xmmreg*,xmmrm64|sae AVX512ER
VRCP28SD xmmreg | mask | z, xmmreg * , xmmrm64 | sae AVX512ER
VRCP28SS         xmmreg|mask|z,xmmreg*,xmmrm32|sae AVX512ER
VRCP28SS xmmreg | mask | z, xmmreg * , xmmrm32 | sae AVX512ER
```

VREDUCEPD	xmmreg|mask|z,xmmrm128|b64,imm8 AVX512VL/DQ
VREDUCEPD	Xmmreg | mask | z， xmmrm128 | b64， imm8 AVX512VL/DQ
VREDUCEPD	ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL/DQ
VREDUCEPD	Ymmreg | mask | z， ymmrm256 | b64， imm8 AVX512VL/DQ
	zmmreg|mask|z,zmmrm512|b64|sae,imm8 AVX512DQ
VREDUCEPD	Zmmreg | mask | z， zmmrm512 | b64 | sae， imm8
VREDUCEPD	AVX512DQ
VREDUCEPS	xmmreg|mask|z,xmmrm128|b32,imm8 AVX512VL/DQ
VREDUCEPS	Xmmreg | mask | z， xmmrm128 | b32， imm8 AVX512VL/DQ
VREDUCEPS	ymmreg|mask|z,ymmrm256|b32,imm8 AVX512VL/DQ
VREDUCEPS	Ymmreg | mask | z， ymmrm256 | b32， imm8 AVX512VL/DQ
	zmmreg|mask|z,zmmrm512|b32|sae,imm8 AVX512DQ
VREDUCEPS	Zmmreg | mask | z， zmmrm512 | b32 | sae， imm8
VREDUCEPS	AVX512DQ
	xmmreg|mask|z,xmmreg*,xmmrm64|sae,imm8 AVX512DQ
VREDUCESD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | sae， imm8
VREDUCESD	AVX512DQ
	xmmreg|mask|z,xmmreg*,xmmrm32|sae,imm8 AVX512DQ
VREDUCESS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | sae， imm8
VREDUCESS	AVX512DQ
VRNDSCALEPD	xmmreg|mask|z,xmmrm128|b64,imm8 AVX512VL
环境保护署	Xmmreg | mask | z， xmmrm128 | b64， imm8 AVX512VL
VRNDSCALEPD	ymmreg|mask|z,ymmrm256|b64,imm8 AVX512VL
环境保护署	Ymmreg | mask | z， ymmrm256 | b64， imm8 AVX512VL
VRNDSCALEPD	zmmreg|mask|z,zmmrm512|b64|sae,imm8 AVX512
环境保护署	Zmmreg | mask | z， zmmrm512 | b64 | sae， imm8 AVX512
VRNDSCALEPS	xmmreg|mask|z,xmmrm128|b32,imm8 AVX512VL
头皮	Xmmreg | mask | z， xmmrm128 | b32， imm8 AVX512VL
VRNDSCALEPS	ymmreg|mask|z,ymmrm256|b32,imm8 AVX512VL
头皮	Ymmreg | mask | z， ymmrm256 | b32， imm8 AVX512VL
VRNDSCALEPS	zmmreg|mask|z,zmmrm512|b32|sae,imm8 AVX512
头皮	Zmmreg | mask | z， zmmrm512 | b32 | sae， imm8 AVX512
	xmmreg|mask|z,xmmreg*,xmmrm64|sae,imm8 AVX512
VRNDSCALESD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | sae， imm8
VRNDSCALESD	AVX512
	xmmreg|mask|z,xmmreg*,xmmrm32|sae,imm8 AVX512
VRNDSCALESS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | sae， imm8
VRNDSCALESS	AVX512
VRSQRT14PD	xmmreg|mask|z,xmmrm128|b64 AVX512VL
VRSQRT14PD	Xmmreg | mask | z， xmmrm128 | b64 AVX512VL
VRSQRT14PD	ymmreg|mask|z,ymmrm256|b64 AVX512VL
VRSQRT14PD	Ymmreg | mask | z， ymmrm256 | b64 AVX512VL
VRSQRT14PD	zmmreg|mask|z,zmmrm512|b64 AVX512
VRSQRT14PD	Zmmrreg | mask | z， zmmrm512 | b64 AVX512
VRSQRT14PS	xmmreg|mask|z,xmmrm128|b32 AVX512VL
VRSQRT14PS	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL
VRSQRT14PS	ymmreg|mask|z,ymmrm256|b32 AVX512VL
VRSQRT14PS	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL
VRSQRT14PS	zmmreg|mask|z,zmmrm512|b32 AVX512
VRSQRT14PS	Zmmrreg | mask | z， zmmrm512 | b32 AVX512

VRSQRT14SD	xmmreg|mask|z,xmmreg*,xmmrm64 AVX512	
VRSQRT14SD	Xmmreg | mask | z， xmmreg * ， xmmrm64 AVX512	
VRSQRT14SS	xmmreg|mask|z,xmmreg*,xmmrm32 AVX512	
VRSQRT14SS	Xmmreg | mask | z， xmmreg * ， xmmrm32 AVX512	
VRSQRT28PD	zmmreg|mask|z,zmmrm512|b64|sae AVX512ER	
VRSQRT28PD	Zmmrreg | mask | z， zmmrm512 | b64 | sae AVX512ER	
VRSQRT28PS	zmmreg|mask|z,zmmrm512|b32|sae AVX512ER	
VRSQRT28PS	Zmmrreg | mask | z， zmmrm512 | b32 | sae AVX512ER	
VRSQRT28SD	xmmreg|mask|z,xmmreg*,xmmrm64|sae AVX512ER	
VRSQRT28SD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | sae AVX512ER	
VRSQRT28SS	xmmreg|mask|z,xmmreg*,xmmrm32|sae AVX512ER	
VRSQRT28SS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | sae AVX512ER	
VSCALEFPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL	
VSCALEFPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL	
VSCALEFPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL	
VSCALEFPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|er AVX512	
VSCALEFPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | er	
VSCALEFPD	AVX512	
VSCALEFPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL	
VSCALEFPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL	
VSCALEFPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL	
VSCALEFPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL	
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|er AVX512	
VSCALEFPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | er	
VSCALEFPS	AVX512	
VSCALEFSD	xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512	
VSCALEFSD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | er AVX512	
VSCALEFSS	xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512	
VSCALEFSS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | er AVX512	
VSCATTERDPD	xmem64|mask,xmmreg	AVX512VL
VSCATTERDPD	Xmem64 | mask， xmmreg	AVX512VL
VSCATTERDPD	xmem64|mask,ymmreg	AVX512VL
VSCATTERDPD	Xmem64 | mask， ymmreg	AVX512VL
VSCATTERDPD	ymem64|mask,zmmreg	AVX512
VSCATTERDPD	Ymem64 | 面具， zmmreg	AVX512
VSCATTERDPS	xmem32|mask,xmmreg	AVX512VL
VSCATTERDPS	Xmem32 | mask， xmmreg	AVX512VL
VSCATTERDPS	ymem32|mask,ymmreg	AVX512VL
VSCATTERDPS	Ymem32 | mask， ymmreg	AVX512VL
VSCATTERDPS	zmem32|mask,zmmreg	AVX512
VSCATTERDPS	Zmem32 | 面具， zmmreg	AVX512
VSCATTERPF0DPD	ymem64|mask	AVX512PF
警察局	Ymem64 | 面具	AVX512PF AVX512PF
VSCATTERPF0DPS	zmem32|mask	AVX512PF
VSCATTERPF0DPS	Zmem32 | 面具	AVX512PF AVX512PF
VSCATTERPF0QPD	zmem64|mask	AVX512PF
VSCATTERPF0QPD	Zmem64 | 面具	AVX512PF AVX512PF
VSCATTERPF0QPS	zmem32|mask	AVX512PF
VSCATTERPF0QPS	Zmem32 | 面具	AVX512PF AVX512PF

VSCATTERPF1DPD	ymem64|mask	AVX512PF
VSCATTERPF1DPD	Ymem64 | 面具	AVX512PF AVX512PF
VSCATTERPF1DPS	zmem32|mask	AVX512PF
VSCATTERPF1DPS	Zmem32 | 面具	AVX512PF AVX512PF
VSCATTERPF1QPD	zmem64|mask	AVX512PF
VSCATTERPF1QPD	Zmem64 | 面具	AVX512PF AVX512PF
VSCATTERPF1QPS	zmem32|mask	AVX512PF
VSCATTERPF1QPS	Zmem32 | 面具	AVX512PF AVX512PF
VSCATTERQPD	xmem64|mask,xmmreg	AVX512VL
VSCATTERQPD	Xmem64 | mask，xmmreg	AVX512VL
VSCATTERQPD	ymem64|mask,ymmreg	AVX512VL
VSCATTERQPD	Ymem64 | 面具，ymmreg	AVX512VL
VSCATTERQPD	zmem64|mask,zmmreg	AVX512
VSCATTERQPD	Zmem64 | mask，zmmreg	AVX512
VSCATTERQPS	xmem32|mask,xmmreg	AVX512VL
VSCATTERQPS	Xmem32 | mask，xmmreg	AVX512VL

228

VSCATTERQPS ymem32|mask,xmmreg AVX512VL
VSCATTERQPS Ymem32 | mask，xmmreg AVX512VL
VSCATTERQPS zmem32|mask,ymmreg AVX512
VSCATTERQPS Zmem32 | 面具，ymmreg AVX512
ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL
VSHUFF32X4 Ymmreg | mask | z，ymmreg *，ymmrm256 | b32，imm8
VSHUFF32X4 AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8 AVX512
VSHUFF32X4 Zmmreg | mask | z，zmmreg *，zmmrm512 | b32，imm8
VSHUFF32X4 AVX512
ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL
VSHUFF64X2 Ymmreg | mask | z，ymmreg *，ymmrm256 | b64，imm8
VSHUFF64X2 AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512
VSHUFF64X2 Zmmreg | mask | z，zmmreg *，zmmrm512 | b64，imm8
VSHUFF64X2 AVX512
ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL
VSHUFI32X4 Ymmreg | mask | z，ymmreg *，ymmrm256 | b32，imm8
VSHUFI32X4 AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8 AVX512
VSHUFI32X4 Zmmreg | mask | z，zmmreg *，zmmrm512 | b32，imm8
VSHUFI32X4 AVX512
ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL
VSHUFI64X2 Ymmreg | mask | z，ymmreg *，ymmrm256 | b64，imm8
VSHUFI64X2 AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512
VSHUFI64X2 Zmmreg | mask | z，zmmreg *，zmmrm512 | b64，imm8
VSHUFI64X2 AVX512
xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VL
VSHUFPD Xmmreg | mask | z，xmmreg *，xmmrm128 | b64，imm8
VSHUFPD AVX512VL
ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL
VSHUFPD Ymmreg | mask | z，ymmreg *，ymmrm256 | b64，imm8
VSHUFPD AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512
VSHUFPD Zmmreg | mask | z，zmmreg *，zmmrm512 | b64，imm8
VSHUFPD AVX512
xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8 AVX512VL
VSHUFPS Xmmreg | mask | z，xmmreg *，xmmrm128 | b32，imm8
VSHUFPS AVX512VL
ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VL
VSHUFPS Ymmreg | mask | z，ymmreg *，ymmrm256 | b32，imm8
VSHUFPS AVX512VL
zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8 AVX512
VSHUFPS Zmmreg | mask | z，zmmreg *，zmmrm512 | b32，imm8
VSHUFPS AVX512
VSQRTPD xmmreg|mask|z,xmmrm128|b64 AVX512VL
VSQRTPD Xmmreg | mask | z，xmmrm128 | b64 AVX512VL
VSQRTPD ymmreg|mask|z,ymmrm256|b64 AVX512VL
VSQRTPD Ymmreg | mask | z，ymmrm256 | b64 AVX512VL

VSQRTPD	zmmreg|mask|z,zmmrm512|b64|er AVX512
VSQRTPD	Zmmrreg | mask | z， zmmrm512 | b64 | er AVX512
VSQRTPS	xmmreg|mask|z,xmmrm128|b32 AVX512VL
VSQRTPS	Xmmreg | mask | z， xmmrm128 | b32 AVX512VL
VSQRTPS	ymmreg|mask|z,ymmrm256|b32 AVX512VL
VSQRTPS	Ymmreg | mask | z， ymmrm256 | b32 AVX512VL
VSQRTPS	zmmreg|mask|z,zmmrm512|b32|er AVX512
VSQRTPS	Zmmrreg | mask | z， zmmrm512 | b32 | er AVX512
VSQRTSD	xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512
VSQRTSD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | er AVX512
VSQRTSS	xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512
VSQRTSS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | er AVX512
VSUBPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VSUBPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VSUBPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VSUBPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64|er AVX512
VSUBPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 | er
VSUBPD	AVX512
VSUBPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VSUBPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VSUBPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VSUBPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32|er AVX512
VSUBPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 | er
VSUBPS	AVX512
VSUBSD	xmmreg|mask|z,xmmreg*,xmmrm64|er AVX512
VSUBSD	Xmmreg | mask | z， xmmreg * ， xmmrm64 | er AVX512
VSUBSS	xmmreg|mask|z,xmmreg*,xmmrm32|er AVX512
VSUBSS	Xmmreg | mask | z， xmmreg * ， xmmrm32 | er AVX512
VUCOMISD	xmmreg,xmmrm64|sae AVX512
VUCOMISD	Xmmrm64 | sae AVX512
VUCOMISS	xmmreg,xmmrm32|sae AVX512
VUCOMISS	Xmmreg， xmmrm32 | sae AVX512
VUNPCKHPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VUNPCKHPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VUNPCKHPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VUNPCKHPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL
VUNPCKHPD	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VUNPCKHPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VUNPCKHPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VUNPCKHPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VUNPCKHPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VUNPCKHPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VUNPCKHPS	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VUNPCKHPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
VUNPCKLPD	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL
VUNPCKLPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VL
VUNPCKLPD	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL
VUNPCKLPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VL

VUNPCKLPD	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512
VUNPCKLPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512
VUNPCKLPS	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL
VUNPCKLPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VL
VUNPCKLPS	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL
VUNPCKLPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VL
VUNPCKLPS	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512
VUNPCKLPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512
	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VL/DQ
VXORPD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64
VXORPD	AVX512VL/DQ
	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VL/DQ
VXORPD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64
VXORPD	AVX512VL/DQ
VXORPD	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512DQ
VXORPD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512DQ
	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VL/DQ
VXORPS	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32
VXORPS	AVX512VL/DQ
	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VL/DQ
VXORPS	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32
VXORPS	AVX512VL/DQ
VXORPS	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512DQ
VXORPS	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512DQ

229

B.1.41 Intel memory protection keys for userspace (PKU aka PKEYs)
B. 1.41 用于用户空间的英特尔内存保护密钥(PKU aka PKEYs)

RDPKRU		X64
RDPKRU		X64
WRPKRU		X64
WRPKRU		X64

B.1.42 Read Processor ID
B. 1.42 读取处理器 ID

RDPID	reg32	NOLONG
RDPID	Reg32	不久
RDPID	reg64	X64
RDPID	Reg64	X64
		X64,UNDOC
RDPID	reg32	X64，
RDPID	Reg32	UNDOC

B.1.43 New memory instructions
B. 1.43 新的内存指令

CLFLUSHOPT	mem	
CLFLUSHOPT	Mem	
CLWB	mem	
CLWB	Mem	
PCOMMIT		UNDOC,OBSOLETE
PCOMMIT		UNDOC，过时了
CLZERO		AMD
CLZERO		AMD
		AMD,ND,NOLONG
CLZERO	reg_ax	AMD，ND，
CLZERO	Reg _ ax	NOLONG
CLZERO	reg_eax	AMD,ND
CLZERO	注册表格	AMD，ND
CLZERO	reg_rax	AMD,ND,X64
CLZERO	Reg _ rax	AMD，ND，X64

B.1.44 Processor trace write
B. 1.44 处理器跟踪写入

PTWRITE	rm32	
PTWRITE	Rm32	
PTWRITE	rm64	X64
PTWRITE	Rm64	X64

B.1.45 Instructions from the Intel Instruction Set Extensions,

B. 1.45 英特尔指令集扩展的指令,

B.1.46 doc 319433−034 May 2018

B. 1.46 doc 319433-034 May 2018

CLDEMOTE	mem	
克莱德莫特	Mem	
MOVDIRI	mem32,reg32	SD
MOVDIRI	Mem32，reg32	标准普尔
MOVDIRI	mem64,reg64	X64

MOVDIRI	Mem64， reg64	X64
MOVDIR64B	reg16,mem512	NOLONG
MOVDIR64B	Reg16， mem512	不久
MOVDIR64B	reg32,mem512	
MOVDIR64B	Reg32， mem512	
MOVDIR64B	reg64,mem512	X64
MOVDIR64B	Reg64， mem512	X64
PCONFIG		
PCONFIG PCONFIG		
TPAUSE	reg32	
暂停	Reg32	
TPAUSE	reg32,reg_edx,reg_eax	ND
暂停	Reg32， reg _ edx， reg _ eax	ND
UMONITOR	reg16	NOLONG
监视器	规例 16	不久
UMONITOR	reg32	
监视器	Reg32	
UMONITOR	reg64	X64
监视器	Reg64	X64
UMWAIT	reg32	
UMWAIT	Reg32	
UMWAIT	reg32,reg_edx,reg_eax	ND
UMWAIT	Reg32， reg _ edx， reg _ eax	ND
WBNOINVD		
WBNOINVD		

B.1.47 Galois field operations (GFNI)

B. 1.47 Galois 野外作业(GFNI)

	xmmreg,xmmrm128,imm8	
GF2P8AFFINEINVQB	Xmmreg， xmmmrm128，	GFNI,SSE
GF2P8AFFINEINVQB	imm8	GFNI SSE
VGF2P8AFFINEINVQB	xmmreg,xmmreg*,xmmrm128,imm8	GFNI,AVX
VGF2P8AFFINEINVQB	xmmreg， xmmreg * ， xmmrm128， imm8	GFNI， AVX
VGF2P8AFFINEINVQB	ymmreg,ymmreg*,ymmrm256,imm8	GFNI,AVX
VGF2P8AFFINEINVQB	ymmreg * ymmreg * ymmrm256 imm8	GFNI AVX
VGF2P8AFFINEINVQB	xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8	AVX512VL,GFNI
VGF2P8AFFINEINVQB	xmmreg | mask | z， xmmreg * ， xmmrm128 | b64， imm8	AVX512VL， GFNI
VGF2P8AFFINEINVQB	ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8	AVX512VL,GFNI
VGF2P8AFFINEINVQB	ymmreg | mask | z， ymmreg * ， ymmrm256 | b64， imm8	AVX512VL， GFNI
VGF2P8AFFINEINVQB	zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8	AVX512,GFNI
VGF2P8AFFINEINVQB	zmmreg | mask | z， zmmreg * ， zmmrm512 | b64， imm8	AVX512， GFNI
	xmmreg,xmmrm128,imm8	
GF2P8AFFINEQB	Xmmreg， xmmmrm128，	GFNI,SSE
GF2P8AFFINEQB	imm8	GFNI SSE
VGF2P8AFFINEQB	xmmreg,xmmreg*,xmmrm128,imm8	GFNI,AVX
VGF2P8AFFINEQB	Xmmreg， xmmreg * ， xmmrm128， imm8	GFNI， AVX
VGF2P8AFFINEQB	ymmreg,ymmreg*,ymmrm256,imm8	GFNI,AVX
VGF2P8AFFINEQB	Ymmreg， ymmreg * ， ymmrm256， imm8	GFNI， AVX

xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VL,GFNI

VGF2P8AFFINEQB Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64， imm8
VGF2P8AFFINEQB AVX512VL， GFNI

230

	ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VL,GFNI	
VGF2P8AFFINEQB	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64， imm8	
VGF2P8AFFINEQB	AVX512VL， GFNI	
	zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512,GFNI	
VGF2P8AFFINEQB	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64， imm8 AVX512，	
VGF2P8AFFINEQB	GFNI	
GF2P8MULB	xmmreg,xmmrm128	GFNI,SSE
GF2P8MULB	Xmmreg， xmmrm128	GFNI SSE
	xmmreg,xmmreg*,xmmrm128	
VGF2P8MULB	Xmmreg， xmmreg * ，	GFNI,AVX
VGF2P8MULB	xmmrm128	GFNI AVX
	ymmreg,ymmreg*,ymmrm256	
VGF2P8MULB	Ymmreg， ymmreg * ，	GFNI,AVX
VGF2P8MULB	ymmrm256	GFNI AVX
VGF2P8MULB	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VL,GFNI	
VGF2P8MULB	Xmmreg | mask | z， xmmreg * ， xmmrm128 AVX512VL， GFNI	
VGF2P8MULB	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VL,GFNI	
VGF2P8MULB	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VL， GFNI	
VGF2P8MULB	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512,GFNI	
VGF2P8MULB	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512， GFNI	

B.1.48 AVX512 Vector Bit Manipulation Instructions 2
B. 1.48 avx512 矢量位操作说明 2

VPCOMPRESSB	mem128|mask,xmmreg	AVX512VBMI2/VL
VPCOMPRESSB	Mem128 | mask， xmmreg	AVX512VBMI2/VL
VPCOMPRESSB	mem256|mask,ymmreg	AVX512VBMI2/VL
VPCOMPRESSB	Mem256 | 面具， ymmreg	AVX512VBMI2/VL
VPCOMPRESSB	mem512|mask,zmmreg	AVX512VBMI2
VPCOMPRESSB	Mem512 | mask， zmmreg	AVX512VBMI2
VPCOMPRESSB	xmmreg|mask|z,xmmreg	AVX512VBMI2/VL
VPCOMPRESSB	| mask | z， xmmreg	AVX512VBMI2/VL
VPCOMPRESSB	ymmreg|mask|z,ymmreg	AVX512VBMI2/VL
VPCOMPRESSB	Ymmreg | mask | z， ymmreg	AVX512VBMI2/VL
VPCOMPRESSB	zmmreg|mask|z,zmmreg	AVX512VBMI2
VPCOMPRESSB	Zmmreg | mask | z， zmmreg	AVX512VBMI2
VPCOMPRESSW	mem128|mask,xmmreg	AVX512VBMI2/VL
VPCOMPRESSW	Mem128 | mask， xmmreg	AVX512VBMI2/VL
VPCOMPRESSW	mem256|mask,ymmreg	AVX512VBMI2/VL
VPCOMPRESSW	Mem256 | 面具， ymmreg	AVX512VBMI2/VL
VPCOMPRESSW	mem512|mask,zmmreg	AVX512VBMI2
VPCOMPRESSW	Mem512 | mask， zmmreg	AVX512VBMI2
VPCOMPRESSW	xmmreg|mask|z,xmmreg	AVX512VBMI2/VL
VPCOMPRESSW	| mask | z， xmmreg	AVX512VBMI2/VL
VPCOMPRESSW	ymmreg|mask|z,ymmreg	AVX512VBMI2/VL
VPCOMPRESSW	Ymmreg | mask | z， ymmreg	AVX512VBMI2/VL
VPCOMPRESSW	zmmreg|mask|z,zmmreg	AVX512VBMI2
VPCOMPRESSW	Zmmreg | mask | z， zmmreg	AVX512VBMI2
VPEXPANDB	mem128|mask,xmmreg	AVX512VBMI2/VL
VPEXPANDB	Mem128 | mask， xmmreg	AVX512VBMI2/VL
VPEXPANDB	mem256|mask,ymmreg	AVX512VBMI2/VL

VPEXPANDB	Mem256 | 面具，ymmreg	AVX512VBMI2/VL
VPEXPANDB	mem512|mask,zmmreg	AVX512VBMI2
VPEXPANDB	Mem512 | mask，zmmreg	AVX512VBMI2
VPEXPANDB	xmmreg|mask|z,xmmreg	AVX512VBMI2/VL
VPEXPANDB	| mask | z，xmmreg	AVX512VBMI2/VL
VPEXPANDB	ymmreg|mask|z,ymmreg	AVX512VBMI2/VL
VPEXPANDB	Ymmreg | mask | z，ymmreg	AVX512VBMI2/VL
VPEXPANDB	zmmreg|mask|z,zmmreg	AVX512VBMI2
VPEXPANDB	Zmmreg | mask | z，zmmreg	AVX512VBMI2
VPEXPANDW	mem128|mask,xmmreg	AVX512VBMI2/VL
Vpexpanw	Mem128 | mask，xmmreg	AVX512VBMI2/VL
VPEXPANDW	mem256|mask,ymmreg	AVX512VBMI2/VL
Vpexpanw	Mem256 | 面具，ymmreg	AVX512VBMI2/VL
VPEXPANDW	mem512|mask,zmmreg	AVX512VBMI2
Vpexpanw	Mem512 | mask，zmmreg	AVX512VBMI2
VPEXPANDW	xmmreg|mask|z,xmmreg	AVX512VBMI2/VL
Vpexpanw	| mask | z，xmmreg	AVX512VBMI2/VL
VPEXPANDW	ymmreg|mask|z,ymmreg	AVX512VBMI2/VL
Vpexpanw	Ymmreg | mask | z，ymmreg	AVX512VBMI2/VL
VPEXPANDW	zmmreg|mask|z,zmmreg	AVX512VBMI2
Vpexpanw	Zmmreg | mask | z，zmmreg	AVX512VBMI2
	xmmreg|mask|z,xmmreg*,xmmrm128,imm8 AVX512VBMI2/VL	
VPSHLDW	Xmmreg | mask | z，xmmreg *，xmmrm128，imm8	
VPSHLDW	AVX512VBMI2/VL	
	ymmreg|mask|z,ymmreg*,ymmrm256,imm8 AVX512VBMI2/VL	
VPSHLDW	Ymmreg | mask | z，ymmreg *，ymmrm256，imm8	
VPSHLDW	AVX512VBMI2/VL	
VPSHLDW	zmmreg|mask|z,zmmreg*,zmmrm512,imm8 AVX512VBMI2	
VPSHLDW	Zmmreg | mask | z，zmmreg *，zmmrm512，imm8 AVX512VBMI2	
	xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8	
VPSHLDD	Xmmreg | mask | z，xmmreg *，xmmrm128 |	AVX512VBMI2/VL
VPSHLDD	b32，imm8	AVX512VBMI2/VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8	
VPSHLDD	Ymmreg | mask | z，ymmreg *，ymmrm256 |	AVX512VBMI2/VL
VPSHLDD	b32，imm8	AVX512VBMI2/VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8	
VPSHLDD	Zmmreg | mask | z，zmmreg *，zmmrm512 |	AVX512VBMI2
VPSHLDD	b32，imm8	AVX512VBMI2
	xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8	
VPSHLDQ	Xmmreg | mask | z，xmmreg *，xmmrm128 |	AVX512VBMI2/VL
VPSHLDQ	b64，imm8	AVX512VBMI2/VL
	ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8	
VPSHLDQ	Ymmreg | mask | z，ymmreg *，ymmrm256 |	AVX512VBMI2/VL
VPSHLDQ	b64，imm8	AVX512VBMI2/VL
	zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8	
VPSHLDQ	Zmmreg | mask | z，zmmreg *，zmmrm512 |	AVX512VBMI2
VPSHLDQ	b64，imm8	AVX512VBMI2
VPSHLDVW	xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VBMI2/VL	
VPSHLDVW	Xmmreg | mask | z，xmmreg *，xmmrm128 AVX512VBMI2/VL	
VPSHLDVW	ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VBMI2/VL	

VPSHLDVW	Ymmreg | mask | z， ymmreg * ， ymmrm256 AVX512VBMI2/VL
VPSHLDVW	zmmreg|mask|z,zmmreg*,zmmrm512 AVX512VBMI2
VPSHLDVW	Zmmreg | mask | z， zmmreg * ， zmmrm512 AVX512VBMI2
VPSHLDVD	xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VBMI2/VL
VPSHLDVD	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b32 AVX512VBMI2/VL
VPSHLDVD	ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VBMI2/VL
VPSHLDVD	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b32 AVX512VBMI2/VL
VPSHLDVD	zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VBMI2
VPSHLDVD	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b32 AVX512VBMI2
VPSHLDVQ	xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VBMI2/VL
VPSHLDVQ	Xmmreg | mask | z， xmmreg * ， xmmrm128 | b64 AVX512VBMI2/VL
VPSHLDVQ	ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VBMI2/VL
VPSHLDVQ	Ymmreg | mask | z， ymmreg * ， ymmrm256 | b64 AVX512VBMI2/VL
VPSHLDVQ	zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512VBMI2
VPSHLDVQ	Zmmreg | mask | z， zmmreg * ， zmmrm512 | b64 AVX512VBMI2
	xmmreg|mask|z,xmmreg*,xmmrm128,imm8 AVX512VBMI2/VL
VPSHRDW	Xmmreg | mask | z， xmmreg * ， xmmrm128， imm8
VPSHRDW	AVX512VBMI2/VL

231

```
VPSHRDW          ymmreg|mask|z,ymmreg*,ymmrm256,imm8 AVX512VBMI2/VL
VPSHRDW ymmreg | mask | z， ymmreg * , ymmrm256， imm8 AVX512VBMI2/VL
VPSHRDW          zmmreg|mask|z,zmmreg*,zmmrm512,imm8 AVX512VBMI2
VPSHRDW zmmreg | mask | z， zmmreg * , zmmrm512， imm8 AVX512VBMI2
VPSHRDD          xmmreg|mask|z,xmmreg*,xmmrm128|b32,imm8 AVX512VBMI2/VL
VPSHRDD xmmreg | mask | z， xmmreg * , xmmrm128 | b32， imm8 AVX512VBMI2/VL
VPSHRDD          ymmreg|mask|z,ymmreg*,ymmrm256|b32,imm8 AVX512VBMI2/VL
VPSHRDD ymmreg | mask | z， ymmreg * , ymmrm256 | b32， imm8 AVX512VBMI2/VL
VPSHRDD          zmmreg|mask|z,zmmreg*,zmmrm512|b32,imm8 AVX512VBMI2
VPSHRDD zmmreg | mask | z， zmmreg * , zmmrm512 | b32， imm8 AVX512VBMI2
VPSHRDQ          xmmreg|mask|z,xmmreg*,xmmrm128|b64,imm8 AVX512VBMI2/VL
VPSHRDQ xmmreg | mask | z， xmmreg * , xmmrm128 | b64， imm8 AVX512VBMI2/VL
VPSHRDQ          ymmreg|mask|z,ymmreg*,ymmrm256|b64,imm8 AVX512VBMI2/VL
VPSHRDQ ymmreg | mask | z， ymmreg * , ymmrm256 | b64， imm8 AVX512VBMI2/VL
VPSHRDQ          zmmreg|mask|z,zmmreg*,zmmrm512|b64,imm8 AVX512VBMI2
VPSHRDQ zmmreg | mask | z， zmmreg * , zmmrm512 | b64， imm8 AVX512VBMI2
VPSHRDVW         xmmreg|mask|z,xmmreg*,xmmrm128 AVX512VBMI2/VL
VPSHRDVW xmmreg | mask | z， xmmreg * , xmmrm128 AVX512VBMI2/VL
VPSHRDVW         ymmreg|mask|z,ymmreg*,ymmrm256 AVX512VBMI2/VL
VPSHRDVW ymmreg | mask | z， ymmreg * , ymmrm256 AVX512VBMI2/VL
VPSHRDVW         zmmreg|mask|z,zmmreg*,zmmrm512 AVX512VBMI2
VPSHRDVW zmmreg | mask | z， zmmreg * , zmmrm512 AVX512VBMI2
VPSHRDVD         xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VBMI2/VL
VPSHRDVD xmmreg | mask | z， xmmreg * , xmmrm128 | b32 AVX512VBMI2/VL
VPSHRDVD         ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VBMI2/VL
VPSHRDVD ymmreg | mask | z， ymmreg * , ymmrm256 | b32 AVX512VBMI2/VL
VPSHRDVD         zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VBMI2
VPSHRDVD zmmreg | mask | z， zmmreg * , zmmrm512 | b32 AVX512VBMI2
VPSHRDVQ         xmmreg|mask|z,xmmreg*,xmmrm128|b64 AVX512VBMI2/VL
VPSHRDVQ xmmreg | mask | z， xmmreg * , xmmrm128 | b64 AVX512VBMI2/VL
VPSHRDVQ         ymmreg|mask|z,ymmreg*,ymmrm256|b64 AVX512VBMI2/VL
VPSHRDVQ ymmreg | mask | z， ymmreg * , ymmrm256 | b64 AVX512VBMI2/VL
VPSHRDVQ         zmmreg|mask|z,zmmreg*,zmmrm512|b64 AVX512VBMI2
VPSHRDVQ zmmreg | mask | z， zmmreg * , zmmrm512 | b64 AVX512VBMI2
```

B.1.49 AVX512 VNNI

B. 1.49 AVX512 VNNI

```
VPDPBUSD         xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VNNI/VL
VPDPBUSD xmmreg | mask | z， xmmreg * , xmmrm128 | b32 AVX512VNNI/VL
VPDPBUSD         ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VNNI/VL
VPDPBUSD ymmreg | mask | z， ymmreg * , ymmrm256 | b32 AVX512VNNI/VL
VPDPBUSD         zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VNNI
VPDPBUSD zmmreg | mask | z， zmmreg * , zmmrm512 | b32 AVX512VNNI
VPDPBUSDS        xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VNNI/VL
VPDPBUSDS xmmreg | mask | z， xmmreg * , xmmrm128 | b32 AVX512VNNI/VL
VPDPBUSDS        ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VNNI/VL
VPDPBUSDS ymmreg | mask | z， ymmreg * , ymmrm256 | b32 AVX512VNNI/VL
VPDPBUSDS        zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VNNI
VPDPBUSDS zmmreg | mask | z， zmmreg * , zmmrm512 | b32 AVX512VNNI
VPDPWSSD         xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VNNI/VL
VPDPWSSD xmmreg | mask | z， xmmreg * , xmmrm128 | b32 AVX512VNNI/VL
VPDPWSSD         ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VNNI/VL
VPDPWSSD ymmreg | mask | z， ymmreg * , ymmrm256 | b32 AVX512VNNI/VL
VPDPWSSD         zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VNNI
VPDPWSSD zmmreg | mask | z， zmmreg * , zmmrm512 | b32 AVX512VNNI
```

```
VPDPWSSDS        xmmreg|mask|z,xmmreg*,xmmrm128|b32 AVX512VNNI/VL
VPDPWSSDS xmmreg | mask | z, xmmreg * , xmmrm128 | b32 AVX512VNNI/VL
VPDPWSSDS        ymmreg|mask|z,ymmreg*,ymmrm256|b32 AVX512VNNI/VL
VPDPWSSDS ymmreg | mask | z, ymmreg * , ymmrm256 | b32 AVX512VNNI/VL
VPDPWSSDS        zmmreg|mask|z,zmmreg*,zmmrm512|b32 AVX512VNNI
VPDPWSSDS zmmreg | mask | z, zmmreg * , zmmrm512 | b32 AVX512VNNI
```

B.1.50 AVX512 Bit Algorithms
B. 1.50 avx512 位算法

	xmmreg|mask|z,xmmrm128	
VPOPCNTB	Xmmreg | mask | z，	AVX512BITALG/VL
VPOPCNTB	xmmrm128	AVX512BITALG/VL
	ymmreg|mask|z,ymmrm256	
VPOPCNTB	Ymmreg | mask | z，	AVX512BITALG/VL
VPOPCNTB	ymmrm256	AVX512BITALG/VL
	zmmreg|mask|z,zmmrm512	
VPOPCNTB	Zmmreg | mask | z，	AVX512BITALG
VPOPCNTB	zmmrm512	AVX512BITALG
	xmmreg|mask|z,xmmrm128	
VPOPCNTW	Xmmreg | mask | z，	AVX512BITALG/VL
VPOPCNTW	xmmrm128	AVX512BITALG/VL
	ymmreg|mask|z,ymmrm256	
VPOPCNTW	Ymmreg | mask | z，	AVX512BITALG/VL
VPOPCNTW	ymmrm256	AVX512BITALG/VL
	zmmreg|mask|z,zmmrm512	
VPOPCNTW	Zmmreg | mask | z，	AVX512BITALG
VPOPCNTW	zmmrm512	AVX512BITALG
	xmmreg|mask|z,xmmrm128	
VPOPCNTD	Xmmreg | mask | z，	AVX512VPOPCNTDQ/VL
VPOPCNTD	xmmrm128	AVX512VPOPCNTDQ/VL
	ymmreg|mask|z,ymmrm256	
VPOPCNTD	Ymmreg | mask | z，	AVX512VPOPCNTDQ/VL
VPOPCNTD	ymmrm256	AVX512VPOPCNTDQ/VL
	zmmreg|mask|z,zmmrm512	
VPOPCNTD	Zmmreg | mask | z，	AVX512VPOPCNTDQ
VPOPCNTD	zmmrm512	AVX512VPOPCNTDQ
	xmmreg|mask|z,xmmrm128	
VPOPCNTQ	Xmmreg | mask | z，	AVX512VPOPCNTDQ/VL
VPOPCNTQ	xmmrm128	AVX512VPOPCNTDQ/VL
	ymmreg|mask|z,ymmrm256	
VPOPCNTQ	Ymmreg | mask | z，	AVX512VPOPCNTDQ/VL
VPOPCNTQ	ymmrm256	AVX512VPOPCNTDQ/VL
	zmmreg|mask|z,zmmrm512	
VPOPCNTQ	Zmmreg | mask | z，	AVX512VPOPCNTDQ
VPOPCNTQ	zmmrm512	AVX512VPOPCNTDQ
VPSHUFBITQMB	kreg|mask,xmmreg,xmmrm128 AVX512BITALG/VL	
VPSHUFBITQMB	Kreg | mask， xmmreg， xmmrm128 AVX512BITALG/VL	
VPSHUFBITQMB	kreg|mask,ymmreg,ymmrm256 AVX512BITALG/VL	
VPSHUFBITQMB	Kreg | mask， ymmreg， ymmrm256 AVX512BITALG/VL	
VPSHUFBITQMB	kreg|mask,zmmreg,zmmrm512 AVX512BITALG	
VPSHUFBITQMB	Kreg | mask， zmmreg， zmmrm512 AVX512BITALG	

B.1.51 AVX512 4−iteration Multiply−Add
B. 1.51 avx5124 次迭代乘加

```
V4FMADDPS        zmmreg|mask|z,zmmreg|rs4,mem AVX5124FMAPS,SO
V4FMADDPS zmmreg | mask | z, zmmreg | rs4, mem AVX5124FMAPS, SO
V4FNMADDPS       zmmreg|mask|z,zmmreg|rs4,mem AVX5124FMAPS,SO
V4FNMADDPS zmmreg | mask | z, zmmreg | rs4, mem AVX5124FMAPS, SO
```

```
V4FMADDSS        zmmreg|mask|z,zmmreg|rs4,mem AVX5124FMAPS,SO
V4FMADDSS zmmreg | mask | z, zmmreg | rs4, mem AVX5124FMAPS, SO
V4FNMADDSS       zmmreg|mask|z,zmmreg|rs4,mem AVX5124FMAPS,SO
V4FNMADDSS zmmreg | mask | z, zmmreg | rs4, mem AVX5124FMAPS, SO
```

B.1.52 AVX512 4−iteration Dot Product
B. 1.52 avx5124 次迭代 Dot 产品

```
V4DPWSSDS        zmmreg|mask|z,zmmreg|rs4,mem AVX5124VNNIW,SO
V4DPWSSDS zmmreg | mask | z, zmmreg | rs4, mem AVX5124VNNIW, SO
V4DPWSSD         zmmreg|mask|z,zmmreg|rs4,mem AVX5124VNNIW,SO
V4DPWSSD zmmreg | mask | z, zmmreg | rs4, mem AVX5124VNNIW, SO
```

B.1.53 Intel Software Guard Extensions (SGX)
1.53 英特尔软件防护扩展(SGX)

```
ENCLS                                          SGX
新加坡交易所
ENCLU                                          SGX
ENCLU SGX
ENCLV                                          SGX
新加坡交易所
```

B.1.54 Systematic names for the hinting nop instructions
B. 1.54 暗示 nop 指令的系统名称

		P6,UNDOC
HINT_NOP0	rm16	P6，
提示: 不	Rm16	UNDOC
		P6,UNDOC
HINT_NOP0	rm32	P6，
提示: 不	Rm32	UNDOC
		X64,UNDOC
HINT_NOP0	rm64	X64，
提示: 不	Rm64	UNDOC
		P6,UNDOC
HINT_NOP1	rm16	P6，
提示 _ nop1	Rm16	UNDOC
		P6,UNDOC
HINT_NOP1	rm32	P6，
提示 _ nop1	Rm32	UNDOC
		X64,UNDOC
HINT_NOP1	rm64	X64，
提示 _ nop1	Rm64	UNDOC
		P6,UNDOC
HINT_NOP2	rm16	P6，
提示: nop2	Rm16	UNDOC
		P6,UNDOC
HINT_NOP2	rm32	P6，
提示: nop2	Rm32	UNDOC
		X64,UNDOC
HINT_NOP2	rm64	X64，
提示: nop2	Rm64	UNDOC
HINT_NOP3	rm16	P6,UNDOC
提示 _nop3	Rm16	P6，

		UNDOC
		P6,UNDOC
HINT_NOP3	rm32	P6，
提示 _nop3	Rm32	UNDOC
		X64,UNDOC
HINT_NOP3	rm64	X64，
提示 _nop3	Rm64	UNDOC
		P6,UNDOC
HINT_NOP4	rm16	P6，
提示: nop4	Rm16	UNDOC
		P6,UNDOC
HINT_NOP4	rm32	P6，
提示: nop4	Rm32	UNDOC
		X64,UNDOC
HINT_NOP4	rm64	X64，
提示: nop4	Rm64	UNDOC
		P6,UNDOC
HINT_NOP5	rm16	P6，
提示 _ nop5	Rm16	UNDOC
		P6,UNDOC
HINT_NOP5	rm32	P6，
提示 _ nop5	Rm32	UNDOC
		X64,UNDOC
HINT_NOP5	rm64	X64，
提示 _ nop5	Rm64	UNDOC
		P6,UNDOC
HINT_NOP6	rm16	P6，
提示，第六条	Rm16	UNDOC
		P6,UNDOC
HINT_NOP6	rm32	P6，
提示，第六条	Rm32	UNDOC
		X64,UNDOC
HINT_NOP6	rm64	X64，
提示，第六条	Rm64	UNDOC
		P6,UNDOC
HINT_NOP7	rm16	P6，
提示 _nop7	Rm16	UNDOC
		P6,UNDOC
HINT_NOP7	rm32	P6，
提示 _nop7	Rm32	UNDOC
		X64,UNDOC
HINT_NOP7	rm64	X64，
提示 _nop7	Rm64	UNDOC
		P6,UNDOC
HINT_NOP8	rm16	P6，
提示: nop8	Rm16	UNDOC
		P6,UNDOC
HINT_NOP8	rm32	P6，
提示: nop8	Rm32	UNDOC
HINT_NOP8	rm64	X64,UNDOC
提示: nop8	Rm64	X64，

		UNDOC
		P6,UNDOC
HINT_NOP9	rm16	P6，
提示 _nop9	Rm16	UNDOC
		P6,UNDOC
HINT_NOP9	rm32	P6，
提示 _nop9	Rm32	UNDOC
		X64,UNDOC
HINT_NOP9	rm64	X64，
提示 _nop9	Rm64	UNDOC
		P6,UNDOC
HINT_NOP10	rm16	P6，
提示: 不	Rm16	UNDOC
		P6,UNDOC
HINT_NOP10	rm32	P6，
提示: 不	Rm32	UNDOC
		X64,UNDOC
HINT_NOP10	rm64	X64，
提示: 不	Rm64	UNDOC
		P6,UNDOC
HINT_NOP11	rm16	P6，
提示 _ nop11	Rm16	UNDOC
		P6,UNDOC
HINT_NOP11	rm32	P6，
提示 _ nop11	Rm32	UNDOC
		X64,UNDOC
HINT_NOP11	rm64	X64，
提示 _ nop11	Rm64	UNDOC
		P6,UNDOC
HINT_NOP12	rm16	P6，
提示，第 12 条	Rm16	UNDOC
		P6,UNDOC
HINT_NOP12	rm32	P6，
提示，第 12 条	Rm32	UNDOC
		X64,UNDOC
HINT_NOP12	rm64	X64，
提示，第 12 条	Rm64	UNDOC
		P6,UNDOC
HINT_NOP13	rm16	P6，
提示，第 13 条	Rm16	UNDOC

233

HINT_NOP13 提示，第 13 条	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP13 提示，第 13 条	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP14 提示 _ nop14	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP14 提示 _ nop14	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP14 提示 _ nop14	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP15 提示，第十五条	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP15 提示，第十五条	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP15 提示，第十五条	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP16 提示 _ nop16	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP16 提示 _ nop16	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP16 提示 _ nop16	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP17 提示，第 17 条	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP17 提示，第 17 条	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP17 提示，第 17 条	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP18 提示 _ nop18	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP18 提示 _ nop18	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP18 提示 _ nop18	rm64 Rm64	X64,UNDOC X64，

		UNDOC
HINT_NOP19	rm16	P6,UNDOC
提示，第十九条	Rm16	P6，UNDOC
HINT_NOP19	rm32	P6,UNDOC
提示，第十九条	Rm32	P6，UNDOC
HINT_NOP19	rm64	X64,UNDOC
提示，第十九条	Rm64	X64，UNDOC
HINT_NOP20	rm16	P6,UNDOC
提示 _ nop20	Rm16	P6，UNDOC
HINT_NOP20	rm32	P6,UNDOC
提示 _ nop20	Rm32	P6，UNDOC
HINT_NOP20	rm64	X64,UNDOC
提示 _ nop20	Rm64	X64，UNDOC
HINT_NOP21	rm16	P6,UNDOC
提示: nop21	Rm16	P6，UNDOC
HINT_NOP21	rm32	P6,UNDOC
提示: nop21	Rm32	P6，UNDOC
HINT_NOP21	rm64	X64,UNDOC
提示: nop21	Rm64	X64，UNDOC
HINT_NOP22	rm16	P6,UNDOC
提示 _nop22	Rm16	P6，UNDOC
HINT_NOP22	rm32	P6,UNDOC
提示 _nop22	Rm32	P6，UNDOC
HINT_NOP22	rm64	X64,UNDOC
提示 _nop22	Rm64	X64，UNDOC
HINT_NOP23	rm16	P6,UNDOC
提示: nop23	Rm16	P6，UNDOC
HINT_NOP23	rm32	P6,UNDOC
提示: nop23	Rm32	P6，UNDOC
HINT_NOP23	rm64	X64,UNDOC
提示: nop23	Rm64	X64，UNDOC
HINT_NOP24	rm16	P6,UNDOC
提示 _ nop24	Rm16	P6，UNDOC
HINT_NOP24	rm32	P6,UNDOC
提示 _ nop24	Rm32	P6，

		UNDOC
HINT_NOP24 提示 _ nop24	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP25 提示，第 25 条	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP25 提示，第 25 条	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP25 提示，第 25 条	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP26 提示 _ nop26	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP26 提示 _ nop26	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP26 提示 _ nop26	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP27 提示 27	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP27 提示 27	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP27 提示 27	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP28 提示 _nop28	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP28 提示 _nop28	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP28 提示 _nop28	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP29 提示，第 29 条	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP29 提示，第 29 条	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP29 提示，第 29 条	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP30 提示 _ nop30	rm16 Rm16	P6,UNDOC P6，

		UNDOC
		P6,UNDOC
HINT_NOP30	rm32	P6，
提示 _ nop30	Rm32	UNDOC
		X64,UNDOC
HINT_NOP30	rm64	X64，
提示 _ nop30	Rm64	UNDOC
		P6,UNDOC
HINT_NOP31	rm16	P6，
提示 _ nop31	Rm16	UNDOC

234

HINT_NOP31 提示 _ nop31	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP31 提示 _ nop31	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP32 提示: nop32	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP32 提示: nop32	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP32 提示: nop32	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP33 提示 _ nop33	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP33 提示 _ nop33	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP33 提示 _ nop33	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP34 提示 34	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP34 提示 34	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP34 提示 34	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP35 提示 _ nop35	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP35 提示 _ nop35	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP35 提示 _ nop35	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP36 提示 36	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP36 提示 36	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP36 提示 36	rm64 Rm64	X64,UNDOC X64，

		UNDOC
		P6,UNDOC
HINT_NOP37	rm16	P6，
提示 _ nop37	Rm16	UNDOC
		P6,UNDOC
HINT_NOP37	rm32	P6，
提示 _ nop37	Rm32	UNDOC
		X64,UNDOC
HINT_NOP37	rm64	X64，
提示 _ nop37	Rm64	UNDOC
		P6,UNDOC
HINT_NOP38	rm16	P6，
提示，第 38 条	Rm16	UNDOC
		P6,UNDOC
HINT_NOP38	rm32	P6，
提示，第 38 条	Rm32	UNDOC
		X64,UNDOC
HINT_NOP38	rm64	X64，
提示，第 38 条	Rm64	UNDOC
		P6,UNDOC
HINT_NOP39	rm16	P6，
提示 _ nop39	Rm16	UNDOC
		P6,UNDOC
HINT_NOP39	rm32	P6，
提示 _ nop39	Rm32	UNDOC
		X64,UNDOC
HINT_NOP39	rm64	X64，
提示 _ nop39	Rm64	UNDOC
		P6,UNDOC
HINT_NOP40	rm16	P6，
提示: nop40	Rm16	UNDOC
		P6,UNDOC
HINT_NOP40	rm32	P6，
提示: nop40	Rm32	UNDOC
		X64,UNDOC
HINT_NOP40	rm64	X64，
提示: nop40	Rm64	UNDOC
		P6,UNDOC
HINT_NOP41	rm16	P6，
提示 _ nop41	Rm16	UNDOC
		P6,UNDOC
HINT_NOP41	rm32	P6，
提示 _ nop41	Rm32	UNDOC
		X64,UNDOC
HINT_NOP41	rm64	X64，
提示 _ nop41	Rm64	UNDOC
		P6,UNDOC
HINT_NOP42	rm16	P6，
提示: nop42	Rm16	UNDOC
HINT_NOP42	rm32	P6,UNDOC
提示: nop42	Rm32	P6，

		UNDOC
HINT_NOP42 提示: nop42	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP43 提示 _ nop43	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP43 提示 _ nop43	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP43 提示 _ nop43	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP44 提示: nop44	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP44 提示: nop44	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP44 提示: nop44	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP45 提示 _ nop45	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP45 提示 _ nop45	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP45 提示 _ nop45	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP46 提示: nop46	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP46 提示: nop46	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP46 提示: nop46	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP47 提示 _ nop47	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP47 提示 _ nop47	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP47 提示 _ nop47	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP48 提示: nop48	rm16 Rm16	P6,UNDOC P6，

		UNDOC
		P6,UNDOC
HINT_NOP48 提示: nop48	rm32 Rm32	P6， UNDOC
		X64,UNDOC
HINT_NOP48 提示: nop48	rm64 Rm64	X64， UNDOC
		P6,UNDOC
HINT_NOP49 提示: nop49	rm16 Rm16	P6， UNDOC

235

HINT_NOP49 提示: nop49	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP49 提示: nop49	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP50 提示 _ nop50	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP50 提示 _ nop50	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP50 提示 _ nop50	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP51 提示: nop51	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP51 提示: nop51	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP51 提示: nop51	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP52 提示 _ nop52	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP52 提示 _ nop52	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP52 提示 _ nop52	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP53 提示: nop53	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP53 提示: nop53	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP53 提示: nop53	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP54 提示 _ nop54	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP54 提示 _ nop54	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP54 提示 _ nop54	rm64 Rm64	X64,UNDOC X64，

		UNDOC
		P6,UNDOC
HINT_NOP55	rm16	P6，
提示: nop55	Rm16	UNDOC
		P6,UNDOC
HINT_NOP55	rm32	P6，
提示: nop55	Rm32	UNDOC
		X64,UNDOC
HINT_NOP55	rm64	X64，
提示: nop55	Rm64	UNDOC
		P6,UNDOC
HINT_NOP56	rm16	P6，
提示 _ nop56	Rm16	UNDOC
		P6,UNDOC
HINT_NOP56	rm32	P6，
提示 _ nop56	Rm32	UNDOC
		X64,UNDOC
HINT_NOP56	rm64	X64，
提示 _ nop56	Rm64	UNDOC
		P6,UNDOC
HINT_NOP57	rm16	P6，
提示: nop57	Rm16	UNDOC
		P6,UNDOC
HINT_NOP57	rm32	P6，
提示: nop57	Rm32	UNDOC
		X64,UNDOC
HINT_NOP57	rm64	X64，
提示: nop57	Rm64	UNDOC
		P6,UNDOC
HINT_NOP58	rm16	P6，
提示 _ nop58	Rm16	UNDOC
		P6,UNDOC
HINT_NOP58	rm32	P6，
提示 _ nop58	Rm32	UNDOC
		X64,UNDOC
HINT_NOP58	rm64	X64，
提示 _ nop58	Rm64	UNDOC
		P6,UNDOC
HINT_NOP59	rm16	P6，
提示: nop59	Rm16	UNDOC
		P6,UNDOC
HINT_NOP59	rm32	P6，
提示: nop59	Rm32	UNDOC
		X64,UNDOC
HINT_NOP59	rm64	X64，
提示: nop59	Rm64	UNDOC
		P6,UNDOC
HINT_NOP60	rm16	P6，
提示 _ nop60	Rm16	UNDOC
HINT_NOP60	rm32	P6,UNDOC
提示 _ nop60	Rm32	P6，

		UNDOC
HINT_NOP60 提示 _ nop60	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP61 提示 61	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP61 提示 61	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP61 提示 61	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP62 提示 _ nop62	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP62 提示 _ nop62	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP62 提示 _ nop62	rm64 Rm64	X64,UNDOC X64， UNDOC
HINT_NOP63 提示 63	rm16 Rm16	P6,UNDOC P6， UNDOC
HINT_NOP63 提示 63	rm32 Rm32	P6,UNDOC P6， UNDOC
HINT_NOP63 提示 63	rm64 Rm64	X64,UNDOC X64， UNDOC

236

Appendix C: NASM Version History
附录 c: NASM 版本历史

C.1 NASM 2 Series
C. 1 NASM 2 系列

The NASM 2 series supports x86−64, and is the production version of NASM since 2007.
NASM 2 系列支持 x86-64，是自 2007 年以来 NASM 的生产版本。

C.1.1 Version 2.14.03
C. 1.1 版本 2.14.03

• Suppress nuisance "`label changed during code generation`" messages after a real error.
•在真正的错误之后取消“在代码生成过程中标签更改”消息。

• Add support for the `merge` and `strings` attributes on ELF sections. See section 7.9.2.
在 ELF 部分添加对合并和字符串属性的支持，参见 7.9.2 节。

C.1.2 Version 2.14.02
C. 1.2 版本 2.14.02

• Fix crash due to multiple errors or warnings during the code generation pass if a list file is specified.
如果指定了一个列表文件，则在代码生成过程中由于多个错误或警告而导致的 Fix 崩溃传递。

C.1.3 Version 2.14.01
C. 1.3 版本 2.14.01

• Create all system−defined macros defore processing command−line given preprocessing directives (`-p`, `-d`, `-u`, `--pragma`, `--before`).
在给定预处理指令(- p,-d,-u,-pragma,-before)之前创建所有系统定义的宏。

• If debugging is enabled, define a `__DEBUG_FORMAT__` predefined macro. See section 4.11.7.
如果启用了调试，请定义 _ _ DEBUG _ format _ _ 预定义的宏。参见第 4.11.7 节。

• Fix an assert for the case in the `obj` format when a `SEG` operator refers to an `EXTERN` symbol declared further down in the code.
当 SEG 操作符引用代码中进一步声明的 EXTERN 符号时，修复一个 obj 格式的断言。

• Fix a corner case in the floating−point code where a binary, octal or hexadecimal floating−point having at least 32, 11, or 8 mantissa digits could produce slightly incorrect results under very specific conditions.
修正了浮点码中的角点情况，在这种情况下，具有至少 32、11 或 8 个尾数的二进制、八进制或十六进制浮点可能会在非常特定的条件下产生稍微不正确的结果。

• Support `-MD` without a filename, for `gcc` compatibility. `-MF` can be used to set the dependencies output filename. See section 2.1.7.
Support-MD 没有文件名，用于 gcc 兼容性。MF 可以用来设置依赖输出文件名。参见第 2.1.7 节。

• Fix `-E` in combination with `-MD`. See section 2.1.21.
Fix-e 与-md 的组合参见第 2.1.21 节。

• Fix missing errors on redefined labels; would cause convergence failure instead which is very slow and not easy to debug.
修复重新定义的标签上缺失的错误; 将导致收敛失败，这是非常缓慢和不容易调试。

• Duplicate definitions of the same label *with the same value* is now explicitly permitted (2.14 would allow it in some circumstances.)
现在明确允许使用相同值对同一标签进行重复定义(2.14 在某些情况下允许这样做)

- Add the option `--no-line` to ignore `%line` directives in the source. See section 2.1.33 and section 4.10.1.

添加-no-line 选项来忽略源代码中的% line 指令。参见第 2.1.33 节和第 4.10.1 节。

C.1.4 Version 2.14
C. 1.4 版本 2.14

- Changed `-I` option semantics by adding a trailing path separator unconditionally.

Changed-i 选项语义，通过无条件件添加尾随路径分隔符。

- Fixed null dereference in corrupted invalid single line macros.

修正了损坏的无效单行宏中的空解引用。

- Fixed division by zero which may happen if source code is malformed.

固定除以零，这可能发生如果源代码是畸形。

- Fixed out of bound access in processing of malformed segment override.

修正了在处理畸形段重写时超出限制的访问。

- Fixed out of bound access in certain `EQU` parsing.

修正了在某些 EQU 解析中超出限制的访问。

- Fixed buffer underflow in float parsing.

固定浮点分析中的缓冲区下流。

- Added `SGX` (Intel Software Guard Extensions) instructions.

增加了 SGX (英特尔软件防护扩展)指令。

- Added `+n` syntax for multiple contiguous registers.

为多个连续寄存器添加 + n 语法。

- Fixed `subsections_via_symbols` for `macho` object format.

固定子节 _ 通过 _ 符号的宏对象格式。

- Added the `--gprefix`, `--gpostfix`, `--lprefix`, and `--lpostfix` command line options, to allow command line base symbol renaming. See section 2.1.28.

添加了-gprefix,-gpostfix,-lprefix 和-lpostfix 命令行选项，允许命令行基本符号重命名。参见第 2.1.28 节。

- Allow label renaming to be specified by `%pragma` in addition to from the command line. See section 6.9.

允许标签重命名除了从命令行指定以外，还要由% 杂注指定。参见第 6.9 节。

- Supported generic `%pragma` namespaces, `output` and `debug`. See section 6.10.

支持通用的% 杂注命名空间，输出和调试。参见第 6.10 节。

- Added the `--pragma` command line option to inject a `%pragma` directive. See section 2.1.29.

添加 -pragma 命令行选项以注入% pragma 指令。参见 2.1.29 节。

- Added the `--before` command line option to accept preprocess statement before input. See section 2.1.30.

添加-before 命令行选项以接受输入前的预处理语句。参见第 2.1.30 节。

- Added `AVX512 VBMI2` (Additional Bit Manipulation), `VNNI` (Vector Neural Network), `BITALG` (Bit Algorithm), and `GFNI` (Galois Field New Instruction) instructions.

增加了 AVX512 VBMI2(附加位操作)、 VNNI (矢量神经网络)、 BITALG (位算法)和 GFNI (Galois Field New Instruction)指令。

- Added the `STATIC` directive for local symbols that should be renamed using global−symbol rules. See section 6.8.

添加了应该使用全局符号规则重命名的本地符号的 STATIC 指令。参见第 6.8 节。

- Allow a symbol to be defined as `EXTERN` and then later overridden as `GLOBAL` or `COMMON`. Furthermore, a symbol declared `EXTERN` and then defined will be treated as `GLOBAL`. See section 6.5.

允许将符号定义为 EXTERN，然后重写为 GLOBAL 或 COMMON。此外，一个符号声明 EXTERN，然后定义将被视为 GLOBAL。参见第 6.5 节。

- The `GLOBAL` directive no longer is required to precede the definition of the symbol.

GLOBAL 指令不再需要放在符号定义之前。

- Support `private_extern` as `macho` specific extension to the `GLOBAL` directive. See section 7.8.5.

支持 private _ extern 作为 GLOBAL 指令的特定扩展。参见第 7.8.5 节。

- Updated `UD0` encoding to match with the specification

更新 ud0 编码以匹配规范

- Added the `--limit-X` command line option to set execution limits. See section 2.1.31.

添加-limit-x 命令行选项来设置执行限制。参见 2.1.31 节。

- Updated the `Codeview` version number to be aligned with `MASM`.

更新了与 MASM 对齐的 Codeview 版本号。

- Added the `--keep-all` command line option to preserve output files. See section 2.1.32.

添加-keep-all 命令行选项以保存输出文件。

- Added the `--include` command line option, an alias to `-P` (section 2.1.18).

添加-include 命令行选项,-p 的别名(第 2.1.18 节)。

- Added the `--help` command line option as an alias to `-h` (section 3.1).

将-help 命令行选项作为别名添加到 -h (第 3.1 节)。

- Added `-W`, `-D`, and `-Q` suffix aliases for `RET` instructions so the operand sizes of these instructions can be encoded without using `o16`, `o32` or `o64`.

为 RET 指令添加 -w、-d 和 -q 后缀别名，因此可以不使用 o16、 o32 或 o64 对这些指令的操作数大小进行编码。

C.1.5 Version 2.13.03
C. 1.5 版本 2.13.03

- Added AVX and AVX512 `VAES*` and `VPCLMULQDQ` instructions.
增加了 AVX 和 AVX512 VAES * 和 VPCLMULQDQ 指令。

- Fixed missing dwarf record in x32 ELF output format.
修正了 x32 ELF 输出格式中缺少的矮小记录。

C.1.6 Version 2.13.02
C. 1.6 版本 2.13.02

- Fix false positive in testing of numeric overflows.
修正数值溢出测试中的假阳性。

- Fix generation of `PEXTRW` instruction.
修复 PEXTRW 指令的生成。

- Fix `smartalign` package which could trigger an error during optimization if the alignment code expanded too much due to optimization of the previous code.
Fix 智能对齐包，如果对齐代码由于前一代码的优化而扩展过多，则可能在优化过程中触发错误。

- Fix a case where negative value in `TIMES` directive causes panic instead of an error.
修正了 TIMES 指令中负值引起恐慌而不是错误的情况。

- Always finalize `.debug_abbrev` section with a null in `dwarf` output format.
在 dwarf 输出格式中使用 null 来终止.debug _ abbrev 部分。

- Support `debug` flag in section attributes for `macho` output format. See section 7.8.1.
在 section 属性中支持调试标志，用于 macho 输出格式。参见第 7.8.1 节。

- Support up to 16 characters in section names for `macho` output format.
为 macho 输出格式支持最多 16 个节名字符。

- Fix missing update of global `BITS` setting if `SECTION` directive specified a bit size using output format−specific extensions (e.g. `USE32` for the `obj` output format.)
如果 SECTION 指令使用特定于输出格式的扩展(例如，对于 obj 输出格式使用 USE32)指定位大小，则修复缺少全局 BITS 设置的更新

- Fix the incorrect generation of VEX−encoded instruction when static mode decorators are specified on scalar instructions, losing the decorators as they require EVEX encoding.

修正了在标量指令中指定静态模式修饰符时 VEX 编码指令的错误生成，丢失了需要 EVEX 编码的修饰符。

- Option `-MW` to quote dependency outputs according to Watcom Make conventions instead of POSIX Make conventions. See section 2.1.11.

Option-MW 根据 watcommake 约定而不是 posixmake 约定引用依赖项输出。参见第 2.1.11 节。

- The `obj` output format now contains embedded dependency file information, unless disabled with

Obj 输出格式现在包含嵌入的依赖文件信息，除非禁用

`%pragma obj nodepend`. See section 7.4.9.

`% pragma obj nodepend` 参见第 `7.4.9` 节。

- Fix generation of dependency lists.

Fix 依赖列表的生成。

- Fix a number of null pointer reference and memory allocation errors.

修正一些空指针引用和内存分配错误。

- Always generate symbol−relative relocations for the `macho64` output format; at least some versions of the XCode/LLVM linker fails for section−relative relocations.

始终为 macho64 输出格式生成符号相对重定位; 至少有些版本的 XCode/LLVM 链接器在部分相对重定位时失败。

C.1.7 Version 2.13.01
C. 1.7 版本 2.13.01

- Fix incorrect output for some types of `FAR` or `SEG` references in the `obj` output format, and possibly other 16−bit output formats.

修正对象输出格式中某些类型的 FAR 或 SEG 引用的不正确输出，可能还有其他 16 位输出格式。

- Fix the address in the list file for an instruction containing a `TIMES` directive.

修正包含 TIMES 指令的指令的列表文件中的地址。

- Fix error with `TIMES` used together with an instruction which can vary in size, e.g. `JMP`.

使用 TIMES 和一个大小不同的指令一起使用的 Fix 错误，例如 JMP。

- Fix breakage on some uses of the `DZ` pseudo−op.

修复 DZ 伪操作的某些用途上的破坏。

C.1.8 Version 2.13
C. 1.8 版本 2.13

- Support the official forms of the `UD0` and `UD1` instructions.

支持 ud0 和 ud1 指令的正式表单。

- Allow self−segment−relative expressions in immediates and displacements, even when combined with an external or otherwise out−of−segment special symbol, e.g.:

允许在直接和位移中使用自分段相对表达式，即使与外部或其他分段外的特殊符号结合在一起，例如:

```
extern foo
外科医生
mov eax,[foo - $ + ebx]                ; Now legal
现在合法了
```

- Handle a 64−bit origin in NDISASM.

在 NDISASM 中处理 64 位原点。

- NASM can now generate sparse output files for relevant output formats, if the underlying operating system supports them.

如果底层操作系统支持的话，NASM 现在可以为相关的输出格式生成稀疏的输出文件。

- The `macho` object format now supports the `subsections_via_symbols` and `no_dead_strip` directives, see section 7.8.4.

Macho 对象格式现在支持子节 _ 通过 _ 符号和没有 _ 死 _ 条指令，见第 7.8.4 节。

- The `macho` object format now supports the `no_dead_strip`, `live_support` and `strip_static_syms` section flags, see section 7.8.1.

Macho 对象格式现在支持 no _ dead _ strip、 live _ support 和 strip _ static _ syms 节标志，参见 7.8.1 节。

- The `macho` object format now supports the `dwarf` debugging format, as required by newer toolchains.

Macho 对象格式现在支持矮人调试格式，这是新的工具链所要求的。

- All warnings can now be suppressed if desired; warnings not otherwise part of any warning class are now considered its own warning class called `other` (e.g. `-w-other`). Furthermore, warning−as−error can now be controlled on a per warning class basis, using the syntax `-w+error=`*warning−class* and its equivalent for all other warning control options. See section 2.1.25 for the command−line options and warning classes and section 6.13 for the `[WARNING]` directive.

如果需要，现在可以抑制所有警告; 不属于任何警告类的警告现在被认为是其自己的警告类称为 other (例如-w-other)。此外，现在可以在每个警告类的基础上控制 warning-as-error，对所有其他警告控制选项使用语法 -w + error = warning-class 及其等价物。关于命令行选项和警告类参见 2.1.25 节，关于[WARNING]指令参见 6.13 节。

- Fix a number of bugs related to AVX−512 decorators.

修正了一些与 AVX-512 装饰器相关的错误。

- Significant improvements to building NASM with Microsoft Visual Studio via `Mkfiles/msvc.mak`. It is now possible to build the full Windows installer binary as long as the necessary prerequisites are installed; see `Mkfiles/README`

通过 Mkfiles/msvc.mak 使用 Microsoft Visual Studio 构建 NASM 的重大改进。现在只要安装了必要的先决条件，就可以构建完整的 windows installer 二进制文件; 参见 Mkfiles/README

- To build NASM with custom modifications (table changes) or from the git tree now requires Perl 5.8 at the very minimum, quite possibly a higher version (Perl 5.24.1 tested.) There is no requirement to have Perl on your system at all if all you want to do is build unmodified NASM from source archives.

- Fix the `{z}` decorator on AVX−512 `VMOVDQ*` instructions.

在 AVX-512 VMOVDQ * 指令上修复{ z } decorator。

- Add new warnings for certain dangerous constructs which never ought to have been allowed. In particular, the `RESB` family of instructions should have been taking a critical expression all along.

为某些不应该被允许的危险结构添加新的警告。特别是 RESB 家族的指令应该一直采用批判性的表达式。

- Fix the EVEX (AVX−512) versions of the `VPBROADCAST`, `VPEXTR`, and `VPINSR` instructions.

修复 VPBROADCAST、 VPEXTR 和 VPINSR 指令的 EVEX (AVX-512)版本。

- Support contracted forms of additional instructions. As a general rule, if an instruction has a non−destructive source immediately after a destination register that isn’t used as an input, NASM supports omitting that source register, using the destination register as that value. This among other things makes it easier to convert SSE code to the equivalent AVX code:

支持附加指令的简化形式。作为一般规则，如果一条指令在目标寄存器之后立即有一个非破坏性的源，而该目标寄存器不用作输入，则 NASM 支持省略该源寄存器，使用目标寄存器作为该值。这使得将 SSE 代码转换为等价的 AVX 代码变得更加容易:

```
addps xmm1,xmm0                    ; SSE   instruction
Addps xmm1，xmm0                   ; SSE   指示
vaddps    ymm1,ymm1,ymm0             AVX   official long form
Vaddps    Ymm1 ymm1 ymm0           ；AVX   正式长表格
vaddps    ymm1,ymm0                  AVX   contracted form
Vaddps    Ymm1 ymm0                ；AVX   合约形式
```

- Fix Codeview malformed compiler version record.

修复 Codeview 格式错误的编译器版本记录。

- Add the `CLWB` and `PCOMMIT` instructions. Note that the `PCOMMIT` instruction has been deprecated and will never be included in a shipping product; it is included for completeness only.

添加 CLWB 和 PCOMMIT 指令。请注意，PCOMMIT 指令已经过时了，将永远不会包含在运输产品中; 它只是为了完整性而包含的。

- Add the `%pragma` preprocessor directive for soft−error directives.

为软错误指令添加% 杂注预处理器指令。

- Add the `RDPID` instruction.

添加 RDPID 指令。

C.1.9 Version 2.12.02
C. 1.9 版本 2.12.02

- Fix preprocessor errors, especially `%error` and `%warning`, inside `%if` statements.

修正预处理器错误，特别是% 错误和% 警告，在% if 语句中。

- Fix relative relocations in 32−bit Mach−O.

修复 32 位 Mach-o 中的相对重定位。

- More Codeview debug format fixes.

更多 Codeview 调试格式修正。

- If the MASM `PTR` keyword is encountered, issue a warning. This is much more likely to indicate a MASM−ism encountered in NASM than it is a valid label. This warning can be suppressed with `-w-ptr`, the `[warning]` directive (see section 2.1.25) or by the macro definition

如果遇到 MASM PTR 关键字，发出警告。这更有可能表明在 NASM 中遇到了 MASM-ism，而不是一个有效的标签。可以使用 -w-ptr、[warning]指令(参见 2.1.25 节)或宏定义来抑制此警告 `%idefine ptr $%?` (see section 4.1.5).

`% 确定 ptr $% ? (见第 4.1.5 节)。`

- When an error or a warning comes from the expansion of a multi−line macro, display the file and line numbers for the expanded macros. Macros defined with `.nolist` do not get displayed.

当多行宏的扩展出现错误或警告时，显示扩展宏的文件和行号。定义的宏。Nolist 不会显示。

- Add macros `ilog2fw()` and `ilog2cw()` to the `ifunc` macro package. See section 5.4.1.

将宏 ilog2fw ()和 ilog2cw ()添加到 ifunc 宏包中，参见第 5.4.1 节。

C.1.10 Version 2.12.01
C. 1.10 版本 2.12.01

- Portability fixes for some platforms.

某些平台的可移植性修复。

- Fix error when not specifying a list file.

修复未指定列表文件时出现的错误。

- Correct the handling of macro−local labels in the Codeview debugging format.

纠正 Codeview 调试格式中宏本地标签的处理。

- Add `CLZERO`, `MONITORX` and `MWAITX` instructions.

添加 CLZERO，MONITORX 和 MWAITX 指令。

C.1.11 Version 2.12
C. 1.11 版本 2.12

- Major fixes to the `macho` backend (section 7.8); earlier versions would produce invalid symbols and relocations on a regular basis.

Maco 后端的主要修复(第 7.8 节) ; 早期版本会定期生成无效的符号和重定位。

- Support for thread−local storage in Mach−O.

在 Mach-o 中支持线程本地存储。

- Support for arbitrary sections in Mach−O.

支持 Mach-o 中的任意部分。

- Fix wrong negative size treated as a big positive value passed into backend causing NASM to crash.

修正错误的负值大小，将其视为传递到后端导致 NASM 崩溃的大正值。

- Fix handling of zero−extending unsigned relocations, we have been printing wrong message and forgot to assign segment with predefined value before passing it into output format.

Fix 处理零扩展无符号重定位，我们已经打印错误的消息，忘记分配预定义的值段，然后将其传递到输出格式。

- Fix potential write of oversized (with size greater than allowed in output format) relative relocations.

Fix 可能写入过大的(大小大于输出格式允许的)相对重定位。

- Portability fixes for building NASM with the LLVM compiler.

用 LLVM 编译器构建 NASM 的可移植性修正。

- Add support of Codeview version 8 (`cv8`) debug format for `win32` and `win64` formats in the `COFF` backend, see section 7.5.3.

在 COFF 后端添加 Codeview version 8(cv8)对 win32 和 win64 格式的调试格式的支持，参见 7.5.3 节。

- Allow 64−bit outputs in 16/32−bit only backends. Unsigned 64−bit relocations are zero−extended from 32−bits with a warning (suppressible via `−w−zext−reloc`); signed 64−bit relocations are an error.

允许 64 位输出在 16/32 位只有后端。Unsigned 64 位重定位从 32 位零扩展，并带有警告(可通过 -w-zext-reloc 抑制) ; 有符号的 64 位重定位是错误的。

- Line numbers in list files now correspond to the lines in the source files, instead of simply being sequential.

列表文件中的行号现在对应于源文件中的行号，而不是简单的顺序。

- There is now an official 64−bit (x64 a.k.a. x86−64) build for Windows.

现在有一个官方的 64 位(x64 也就是 x86-64)的 Windows 版本。

C.1.12 Version 2.11.09
C. 1.12 版本 2.11.09

- Fix potential stack overwrite in `macho32` backend.

修正 macho32 后端可能的堆栈覆盖问题。

- Fix relocation records in `macho64` backend.

修复 macho64 后端的重定位记录。

- Fix symbol lookup computation in `macho64` backend.

修正了 macho64 后端的符号查找计算。

- Adjust `.symtab` and `.rela.text` sections alignments to 8 bytes in `elf64` backed.

调整.symtab 和.rela.text 部分在 elf64 中的对齐方式为 8 字节。

- Fix section length computation in `bin` backend which leaded in incorrect relocation records.

Fix 区段长度计算在仓库后端导致不正确的重定位记录。

C.1.13 Version 2.11.08
C. 1.13 版本 2.11.08

- Fix section length computation in `bin` backend which leaded in incorrect relocation records.

Fix 区段长度计算在仓库后端导致不正确的重定位记录。

- Add a warning for numeric preprocessor definitions passed via command line which might have unexpected results otherwise.

为通过命令行传递的数字预处理器定义添加一个警告，否则可能会有意想不到的结果。

- Add ability to specify a module name record in `rdoff` linker with `−mn` option.

Add 能够在具有 -mn 选项的 rdoff 链接器中指定模块名记录。

- Increase label length capacity up to 256 bytes in `rdoff` backend for FreePascal sake, which tends to generate very long labels for procedures.

为了 FreePascal 的缘故，在 rdoff 后端增加标签长度容量至 256 字节，这往往会为过程生成非常长的标签。

• Fix segmentation failure when rip addressing is used in `macho64` backend.

在 macho64 后端使用 rip 寻址时修复分段失败。

• Fix access on out of memory when handling strings with a single grave. We have sixed similar problem in previous release but not all cases were covered.

修正了单坟墓处理字符串时内存不足的问题。我们在之前的版本中已经解决了六个类似的问题，但是并不是所有的情况都包括在内。

• Fix NULL dereference in disassembled on `BND` instruction.

修正在 BND 指令上反汇编的 NULL 解引用。

C.1.14 Version 2.11.07

C. 1.14 版本 2.11.07

• Fix 256 bit `VMOVNTPS` instruction.

修正 256 位 VMOVNTPS 指令。

• Fix `-MD` option handling, which was rather broken in previous release changing command line api.

Fix-MD 选项处理，在以前更改命令行 api 的版本中被破坏了。

• Fix access to unitialized space when handling strings with a single grave.

修正在处理带有单个坟墓的字符串时对单一空间的访问。

• Fix nil dereference in handling memory reference parsing.

Fix 处理内存引用解析时的 nil 解引用。

C.1.15 Version 2.11.06

C. 1.15 版本 2.11.06

• Update AVX512 instructions based on the Extension Reference (319433−021 Sept 2014).

基于扩展参考更新 avx512 指令(2014 年 9 月 319433-021)。

• Fix the behavior of `-MF` and `-MD` options (Bugzilla 3392280)

修正 -MF 和 -MD 选项的行为(Bugzilla 3392280)

- Updated Win32 Makefile to fix issue with build

•更新 win32makefile 以修复构建问题

C.1.16 Version 2.11.05
C. 1.16 版本 2.11.05

- Add `--v` as an alias for `-v` (see section 2.1.26), for command−line compatibility with Yasm.

Add-v 作为 -v 的别名(参见 2.1.26 节)，用于命令行与 Yasm 的兼容性。

- Fix a bug introduced in 2.11.03 whereby certain instructions would contain multiple REX prefixes, and thus be corrupt.

修正了在 2.11.03 中引入的一个 bug，在这个 bug 中，某些指令会包含多个 REX 前缀，因此会被破坏。

C.1.17 Version 2.11.04
C. 1.17 版本 2.11.04

- Removed an invalid error checking code. Sometimes a memref only with a displacement can also set an evex flag. For example:

删除了无效的错误检查代码。有时候一个只有位移的 memref 也可以设置一个 evex 标志。例如:

```
vmovdqu32 [0xabcd]{k1}, zmm0
[0xabcd ]{ k1} , zmm0
```

- Fixed a bug in disassembler that EVEX.L'L vector length was not matched when EVEX.b was set because it was simply considered as EVEC.RC. Separated EVEX.L'L case from EVEX.RC which is ignored in matching.

修正了反汇编程序中 EVEX.L'l 矢量长度在 EVEX.b 设置时不匹配的错误，因为它被简单地认为是 EVEC.RC。从 EVEX.RC 中分离的 EVEX.L'l 案例在匹配中被忽略。

C.1.18 Version 2.11.03
C. 1.18 版本 2.11.03

- Fix a bug there REX prefixes were missing on instructions inside a `TIMES` statement.

修复一个错误 REX 前缀在时代周刊的一份声明中的指令中丢失。

C.1.19 Version 2.11.02
C. 1.19 版本 2.11.02

- Add the `XSAVEC`, `XSAVES` and `XRSTORS` family instructions.

添加 XSAVEC、 xsave 和 XRSTORS 家族指令。

- Add the `CLFLUSHOPT` instruction.

添加 CLFLUSHOPT 指令。

C.1.20 Version 2.11.01
C. 1.20 版本 2.11.01

- Allow instructions which implicitly uses `XMM0` (`VBLENDVPD`, `VBLENDVPS`, `PBLENDVB` and `SHA256RNDS2`) to be specified without an explicit `xmm0` on the assembly line. In other words, the following two lines produce the same output:

允许在装配线上指定隐式使用 XMM0(VBLENDVPD、 VBLENDVPS、 PBLENDVB 和 SHA256RNDS2)的指令而不显式使用 XMM0。换句话说，下面两行产生相同的输出:

```
vblendvpd xmm2,xmm1,xmm0       ; Last operand is fixed xmm0
Vblendvpd xmm2, xmm1, xmm0; Last 操作数是固定的
vblendvpd xmm2,xmm1            ; Implicit xmm0 omitted
Vblendvpd xmm2, xmm1; 隐式 xmm0 省略
```

- In the ELF backends, don't crash the assembler if `section align` is specified without a value.

在 ELF 后端，如果节对齐没有指定值，不要崩溃汇编程序。

C.1.21 Version 2.11

C. 1.21 版本 2.11

- Add support for the Intel AVX−512 instruction set:

增加对 Intel AVX-512 指令集的支持:

- 16 new, 512−bit SIMD registers. Total 32 (ZMM0 ~ ZMM31)

•16 个新的 512 位 SIMD 寄存器. Total 32(ZMM0 ~ ZMM31)

- 8 new opmask registers (K0 ~ K7). One of 7 registers (K1 ~ K7) can be used as an opmask for conditional execution.

8 个新的 opmask 寄存器(K0 ~ K7)。7 个寄存器中的一个(K1 ~ K7)可以用作条件执行的 opmask。

- A new EVEX encoding prefix. EVEX is based on VEX and provides more capabilities: opmasks, broadcasting, embedded rounding and compressed displacements.

一个新的 EVEX 编码前缀。EVEX 基于 VEX，提供更多的功能: opmask，广播，嵌入式舍入和压缩位移。

```
 - opmask
 Opmask
     VDIVPD zmm0{k1}{z}, zmm1, zmm3  ; conditional vector
                                     operation ; using opmask k1.
     VDIVPD zmm0{ k1}{ z } , zmm1, zmm3; 条件向量运算; 使用 opmask
                                     k1。
                                     ; {z} is for zero-masking
                                     ; { z }用于零掩码
 - broadcasting
 广播
     VDIVPS zmm4, zmm5, [rbx]{1to16} ; load single-precision float and
     VDIVPS zmm4, zmm5, [ rbx ]{1to16} ; 加载单精度浮点数和
                                     ; replicate it 16 times. 32 * 16 = 512
                                                 重复 16 次. 32 * 16 = 512
 - embedded rounding
 - 嵌入式舍入
     VCVTSI2SD xmm6, xmm7, {rz-sae}, rax ; round toward zero. note that it
     VCVTSI2SD xmm6, xmm7, { rz-sae } , rax; 向零舍入。注意
```

```
                                                    ; is used as if a separate operand.
                                                    ; 如同单独的操作数一样使用。
                                                    ; it comes after the last SIMD operand
                                                    它位于最后一个 SIMD 操作数之后
```

- Add support for `ZWORD` (512 bits), `DZ` and `RESZ`.

添加对 ZWORD (512 位)、 DZ 和 RESZ 的支持。

- Add support for the MPX and SHA instruction sets.

添加对 MPX 和 SHA 指令集的支持。

- Better handling of section redefinition.

更好地处理部分重新定义。

- Generate manpages when running `'make dist'`.

当运行“ make dist”时生成 manpages。

- Handle all token chains in mmacro params range.

处理 mmacro 参数范围内的所有令牌链。

- Support split [base,index] effective address:

支持拆分[基础，索引]有效地址:

```
mov eax,[eax+8,ecx*4] ; eax=base, ecx=index, 4=scale, 8=disp
```

This is expected to be most useful for the MPX instructions.

```
Mov eax， [ eax + 8，ecx * 4] ; eax = base，ecx = index，4 =
scale，8 = disp.
```

- Support `BND` prefix for branch instructions (for MPX).

支持分支指令的 BND 前缀(对于 MPX)。

- The `DEFAULT` directive can now take `BND` and `NOBND` options to indicate whether all relevant branches should be getting `BND` prefixes. This is expected to be the normal for use in MPX code.

DEFAULT 指令现在可以使用 BND 和 NOBND 选项来指示是否所有相关的分支都应该使用 BND 前缀。这在 MPX 代码中应该是正常的。

- Add `{evex}`, `{vex3}` and `{vex2}` instruction prefixes to have NASM encode the corresponding instruction, if possible, with an EVEX, 3−byte VEX, or 2−byte VEX prefix, respectively.

添加{ EVEX }、{ vex3}和{ vex2}指令前缀，使 NASM 分别使用 EVEX、3 字节 VEX 或 2 字节 VEX 前缀对相应的指令进行编码(如果可能的话)。

- Support for section names longer than 8 bytes in Win32/Win64 COFF.

在 Win32/win64coff 中支持超过 8 字节的节名。

- The `NOSPLIT` directive by itself no longer forces a single register to become an index register, unless it has an explicit multiplier.

NOSPLIT 指令本身不再强制一个寄存器成为索引寄存器，除非它有一个显式的乘数。

```
mov eax,[nosplit eax]       ; eax as base register
作为基本寄存器
mov eax,[nosplit eax*1]     ; eax as index register
Mov eax， [ nosplit eax * 1] ; eax 作为索引寄存器
```

C.1.22 Version 2.10.09
C. 1.22 版本 2.10.09

- Pregenerate man pages.

生成手册页。

C.1.23 Version 2.10.08
C. 1.23 版本 2.10.08

- Fix `VMOVNTDQA`, `MOVNTDQA` and `MOVLPD` instructions.

修正 VMOVNTDQA，MOVNTDQA 和 MOVLPD 指令。

- Fix collision for `VGATHERQPS`, `VPGATHERQD` instructions.

修正 vgather qps 的冲突，vpgather qd 指令。

- Fix `VPMOVSXBQ`, `VGATHERQPD`, `VSPLLW` instructions.

修正 VPMOVSXBQ，vgather qpd，VSPLLW 指令。

- Add a bunch of AMD TBM instructions.

添加一系列 AMD TBM 指令。

- Fix potential stack overwrite in numbers conversion.

修正数字转换中可能的堆栈覆盖问题。

- Allow byte size in `PREFETCHTx` instructions.

在 PREFETCHTx 指令中允许字节大小。

- Make manual pages up to date.

更新手册页面。

- Make `F3` and `F2` SSE prefixes to override `66`.

使 f3 和 F2 SSE 前缀覆盖 66。

- Support of AMD SVM instructions in 32 bit mode.

在 32 位模式下支持 AMD SVM 指令。

- Fix near offsets code generation for `JMP`, `CALL` instrictions in long mode.

修正了在长模式下为 JMP，CALL 约束生成近偏移量代码的问题。

- Fix preprocessor parse regression when id is expanding to a whitespace.

Fix 预处理器当 id 扩展到一个空格时解析回归。

C.1.24 Version 2.10.07
C. 1.24 版本 2.10.07

- Fix line continuation parsing being broken in previous version.

修复了前一版本中断的行继续解析。

C.1.25 Version 2.10.06
C. 1.25 版本 2.10.06

- Always quote the dependency source names when using the automatic dependency generation options.

在使用自动依赖项生成选项时，始终引用依赖项源名称。

- If no dependency target name is specified via the `-MT` or `-MQ` options, quote the default output name.

如果没有通过 -MT 或 -MQ 选项指定依赖目标名称，则引用默认输出名称。

- Fix assembly of shift operations in `CPU 8086` mode.

修正了在 CPU 8086 模式下移位操作的汇编。

- Fix incorrect generation of explicit immediate byte for shift by 1 under certain circumstances.

修正在特定情况下移位时显式立即字节的错误生成。

- Fix assembly of the `VPCMPGTQ` instruction.

固定装配的 VPCMPGTQ 指令。

- Fix RIP−relative relocations in the `macho64` backend.

修复 RIP —— macho64 后端的相对重定位。

C.1.26 Version 2.10.05
C. 1.26 版本 2.10.05

- Add the `CLAC` and `STAC` instructions.

•添加 CLAC 和 STAC 说明。

C.1.27 Version 2.10.04
C. 1.27 版本 2.10.04

- Add back the inadvertently deleted 256−bit version of the `VORPD` instruction.

添加回无意中删除的 256 位版本的 VORPD 指令。

- Correct disassembly of instructions starting with byte `82` hex.

以字节 82 十六进制开始的指令的正确反汇编。

- Fix corner cases in token pasting, for example:

修正令牌粘贴中的角点情况，例如:

```
%define N 1e%++%+ 5
% 定义 n 1e% + +% + 5
        dd N, 1e+5
        Dd n，1e + 5
```

C.1.28 Version 2.10.03
C. 1.28 版本 2.10.03

- Correct the assembly of the instruction:

纠正指令的组合:

```
XRELEASE MOV [absolute],AL
[绝对]
```

Previous versions would incorrectly generate `F3 A2` for this instruction and issue a warning; correct behavior is to emit `F3 88 05`.
以前的版本将不正确地为这个指令生成 F3 a2 并发出警告; 正确的行为是发出 f38805。

C.1.29 Version 2.10.02
C. 1.29 版本 2.10.02

- Add the `ifunc` macro package with integer functions, currently only integer logarithms. See section 5.4.

添加带整数函数的 ifunc 宏包，目前只有整数对数。参见第 5.4 节。

- Add the `RDSEED`, `ADCX` and `ADOX` instructions.

添加 RDSEED、 ADCX 和 ADOX 指令。

C.1.30 Version 2.10.01
C. 1.30 版本 2.10.01

- Add missing VPMOVMSKB instruction with reg32, ymmreg operands.

使用 reg32，ymmreg 操作数添加缺少的 VPMOVMSKB 指令。

C.1.31 Version 2.10
C. 1.31 版本 2.10

- When optimization is enabled, `mov r64,imm` now optimizes to the shortest form possible between:

•当启用优化时，mov r64，imm 现在优化到尽可能短的形式:

```
mov r32,imm32                              bytes
Mov r32，imm32                        ;   5 字节
mov
动
起   r64,imm32                              bytes
来   R64，imm32                        ;   7 字节
mov
动
起   r64,imm64                         ; 10 bytes
来   R64，imm64                           10 字节
```

To force a specific form, use the `STRICT` keyword, see section 3.7.
要强制使用特定的表单，请使用 STRICT 关键字，参见第 3.7 节。

- Add support for the Intel AVX2 instruction set.

添加对 Intel avx2 指令集的支持。

- Add support for Bit Manipulation Instructions 1 and 2.

添加对位操作指令 1 和 2 的支持。

- Add support for Intel Transactional Synchronization Extensions (TSX).

添加对 Intel Transactional Synchronization Extensions (TSX)的支持。

- Add support for x32 ELF (32−bit ELF with the CPU in 64−bit mode.) See section 7.9.

添加对 x32 ELF 的支持(CPU 在 64 位模式下的 32 位 ELF)参见第 7.9 节。

- Add support for bigendian UTF−16 and UTF−32. See section 3.4.5.

添加对二进制 UTF-16 和 UTF-32 的支持，参见第 3.4.5 节。

C.1.32 Version 2.09.10
C. 1.32 Version 2.09.10

- Fix up NSIS script to protect uninstaller against registry keys absence or corruption. It brings in a few additional questions to a user during deinstallation procedure but still it is better than unpredictable file removal.

修正 NSIS 脚本以保护卸载程序不受注册表项缺失或损坏的影响。它在卸载过程中给用户带来了一些额外的问题，但仍然比不可预知的文件删除要好。

C.1.33 Version 2.09.09
C. 1.33 Version 2.09.09

- Fix initialization of section attributes of `bin` output format.

Fix 初始化 bin 输出格式的 section 属性。

- Fix `mach64` output format bug that crashes NASM due to NULL symbols.

Fix mach64 输出格式错误，由于 NULL 符号导致 NASM 崩溃。

C.1.34 Version 2.09.08
C. 1.34 版本 2.09.08

- Fix `__OUTPUT_FORMAT__` assignment when output driver alias is used. For example when

当使用输出驱动别名时，Fix _ _ OUTPUT _ format _ 赋值

`-f elf` is used `__OUTPUT_FORMAT__` must be set to `elf`, if `-f elf32` is used
如果使用 `-felf32`，则必须将 `_ _ OUTPUT _ format _ _` 设置为 `elf`
`__OUTPUT_FORMAT__` must be assigned accordingly, i.e. to `elf32`. The rule applies to all output driver aliases. See section 4.11.6.
`_ _ OUTPUT _ format _ _` 必须相应地分配，即分配给 `elf32`。该规则适用于所有输出驱动程序别名。参见第 `4.11.6` 节。

C.1.35 Version 2.09.07
C. 1.35 版本 2.09.07

- Fix attempts to close same file several times when `-a` option is used.

Fix 在使用 -a 选项时多次尝试关闭同一个文件。

- Fixes for VEXTRACTF128, VMASKMOVPS encoding.

修复 VEXTRACTF128，VMASKMOVPS 编码。

C.1.36 Version 2.09.06
C. 1.36 版本 2.09.06

- Fix missed section attribute initialization in `bin` output target.

修复在 bin 输出目标中错过的 section 属性初始化。

C.1.37 Version 2.09.05
C. 1.37 Version 2.09.05

- Fix arguments encoding for VPEXTRW instruction.

修正 VPEXTRW 指令的参数编码。

- Remove invalid form of VPEXTRW instruction.

删除无效形式的 VPEXTRW 指令。

- Add `VLDDQU` as alias for `VLDQQU` to match specification.

添加 VLDDQU 作为 VLDQQU 的别名以符合规范。

C.1.38 Version 2.09.04
C. 1.38 Version 2.09.04

- Fix incorrect labels offset for VEX intructions.

修正 VEX 指令的不正确标签偏移。

- Eliminate bogus warning on implicit operand size override.

消除隐式操作数大小覆盖的虚假警告。

- `%if` term could not handle 64 bit numbers.

如果术语无法处理 64 位数字，则为% 。

- The COFF backend was limiting relocations number to 16 bits even if in real there were a way more relocations.

COFF 后端将重定位数量限制在 16 位，即使实际上有更多的重定位。

C.1.39 Version 2.09.03
C. 1.39 版本 2.09.03

- Print `%macro` name inside `%rep` blocks on error.

错误时打印% rep 中的宏名称。

- Fix preprocessor expansion behaviour. It happened sometime too early and sometime simply wrong. Move behaviour back to the origins (down to NASM 2.05.01).

修正预处理器扩展行为。这种情况有时发生得太早，有时则是完全错误的。让行为回到起源(下降到 NASM 2.05.01)。

- Fix unitialized data dereference on OMF output format.

Fix 在 OMF 输出格式上的统一数据解引用。

- Issue warning on unterminated `%{` construct.

对未终止的% {构造发出警告。

- Fix for documentation typo.

修正文档输入错误。

245

C.1.40 Version 2.09.02
C. 1.40 版本 2.09.02

- Fix reversed tokens when `%deftok` produces more than one output token.

当% deftok 生成多个输出令牌时修复反向令牌。

- Fix segmentation fault on disassembling some VEX instructions.

Fix 反汇编某些 VEX 指令时的内存区段错误。

- Missing `%endif` did not always cause error.

丢失% endif 并不总是导致错误。

- Fix typo in documentation.

修正文档中的拼写错误。

- Compound context local preprocessor single line macro identifiers were not expanded early enough and as result lead to unresolved symbols.

复合上下文本地预处理器单行宏标识符没有得到足够早的扩展，从而导致未解决的符号。

C.1.41 Version 2.09.01
C. 1.41 版本 2.09.01

- Fix NULL dereference on missed %deftok second parameter.

对错过的第二个参数% deftok 进行 Fix NULL 解引用。

- Fix NULL dereference on invalid %substr parameters.

Fix NULL 解引用无效的% substr 参数。

C.1.42 Version 2.09
C. 1.42 版本 2.09

- Fixed assignment the magnitude of `%rep` counter. It is limited to 62 bits now.

修正了百分比计数器的大小，现在限制为 62 位。

- Fixed NULL dereference if argument of `%strlen` resolves to whitespace. For example if nonexistent macro parameter is used.

如果% strlen 的参数解析为空格，则解引用为 NULL。例如，如果使用了不存在的宏参数。

- `%ifenv`, `%elifenv`, `%ifnenv`, and `%elifnenv` directives introduced. See section 4.4.9.

`% ifenv,% elifenv,% ifnenv`，和`% elifnenv` 指令。参见第 `4.4.9` 节。

- Fixed NULL dereference if environment variable is missed.

如果遗漏环境变量，则修正 NULL 解引用。

- Updates of new AVX v7 Intel instructions.

更新新的 AVX v7 英特尔指令。

- `PUSH imm32` is now officially documented.

`PUSH imm32` 现在正式记录在案。

- Fix for encoding the LFS, LGS and LSS in 64−bit mode.

Fix 用于在 64 位模式下编码 LFS、 LGS 和 LSS。

- Fixes for compatibility with OpenWatcom compiler and DOS 8.3 file format limitation.

修正与 OpenWatcom 编译器兼容性和 dos8.3 文件格式限制。

- Macros parameters range expansion introduced. See section 4.3.4.

Macros 参数范围扩展引入。参见第 4.3.4 节。

- Backward compatibility on expanging of local sigle macros restored.

本地单个宏展开时的向下兼容恢复。

- 8 bit relocations for `elf` and `bin` output formats are introduced.

引入了 elf 和 bin 输出格式的 8 位重定位。

- Short intersegment jumps are permitted now.
现在允许短段间跳跃。

- An alignment more than 64 bytes are allowed for `win32`, `win64` output formats.
Win32，win64 输出格式允许超过 64 字节的对齐。

- `SECTALIGN` directive introduced. See section 4.11.14.
引入 `SECTALIGN` 指令，参见第 `4.11.14` 节。

- `nojmp` option introduced in `smartalign` package. See section 5.2.
`Nojmp` 选项被引入到 `smartalign` 软件包中。参见第 `5.2` 节。

- Short aliases `win`, `elf` and `macho` for output formats are introduced. Each stands for `win32`, `elf32` and `macho32` accordingly.
介绍了短别名 win、 elf 和 macho 的输出格式，它们分别代表 win32、 elf32 和 macho32。

- Faster handling of missing directives implemented.
实现了对缺失指令的更快处理。

- Various small improvements in documentation.
对文档的各种小改进。

- No hang anymore if unable to open malloc.log file.
如果无法打开 malloc.log 文件，则不再挂起。

- The environments without vsnprintf function are able to build nasm again.
没有 vsnprintf 函数的环境可以重新构建 nasm。

- AMD LWP instructions updated.
AMD LWP 指令更新。

- Tighten EA checks. We warn a user if there overflow in EA addressing.
加强 EA 检查。如果 EA 地址有溢出，我们会警告用户。

- Make `-Ox` the default optimization level. For the legacy behavior, specify `-O0` explicitly. See section 2.1.23.
Make-Ox 是默认的优化级别。对于遗留行为，明确指定 -O0。参见 2.1.23 节。

- Environment variables read with `%!` or tested with `%ifenv` can now contain non−identifier characters if surrounded by quotes. See section 4.10.2.

环境变量读取% ！ 或者使用% ifenv 进行测试，如果被引号包围，则可以包含非标识符字符。参见第 4.10.2 节。

- Add a new standard macro package `%use fp` for floating−point convenience macros. See section 5.3.

为浮点方便的宏添加一个新的标准宏包% 使用 fp。参见第 5.3 节。

C.1.43 Version 2.08.02

C. 1.43 Version 2.08.02

- Fix crash under certain circumstances when using the `%+` operator.

•在某些情况下使用% + 运算符时修复崩溃。

C.1.44 Version 2.08.01

C. 1.44 Version 2.08.01

- Fix the `%use` statement, which was broken in 2.08.

修复在 2.08 中被破坏的% use 语句。

C.1.45 Version 2.08

C. 1.45 Version 2.08

- A number of enhancements/fixes in macros area.

宏领域的许多增强/修复。

- Support for converting strings to tokens. See section 4.1.9.

支持将字符串转换为令牌参见第 4.1.9 节。

- Fuzzy operand size logic introduced.

引入了模糊操作数大小逻辑。

- Fix COFF stack overrun on too long export identifiers.

修正 COFF 堆栈在过长的导出标识符上的溢出问题。

- Fix Macho−O alignment bug.

修正 Macho-o 对齐错误。

- Fix crashes with –fwin32 on file with many exports.

Fix 崩溃与 -fwin32 文件与许多导出。

- Fix stack overrun for too long [DEBUG id].

修复堆栈溢出时间过长[DEBUG id]。

- Fix incorrect sbyte usage in IMUL (hit only if optimization flag passed).

修正 IMUL 中不正确的字节使用(仅当优化标志通过时才触发)。

- Append ending token for `.stabs` records in the ELF output format.

以 ELF 输出格式为.stab 记录附加结束令牌。

- New NSIS script which uses ModernUI and MultiUser approach.

使用 ModernUI 和 MultiUser 方法的新 NSIS 脚本。

- Visual Studio 2008 NASM integration (rules file).

Visualstudio2008nasm 集成(规则文件)。

- Warn a user if a constant is too long (and as result will be stripped).

如果一个常量太长，警告用户(结果会被删除)。

- The obsoleted pre−XOP AMD SSE5 instruction set which was never actualized was removed.

过时的前 XOP AMD sse5 指令集从未实现被删除。

- Fix stack overrun on too long error file name passed from the command line.

修复从命令行传来的过长错误文件名的堆栈溢出。

- Bind symbols to the .text section by default (ie in case if SECTION directive was omitted) in the ELF output format.

将符号绑定到。默认情况下(如果 SECTION 指令被省略)使用 ELF 输出格式。

- Fix sync points array index wrapping.

Fix 同步点数组索引包装。

- A few fixes for FMA4 and XOP instruction templates.

Fma4 和 XOP 指令模板的一些修复。

- Add AMD Lightweight Profiling (LWP) instructions.

添加 AMD Lightweight Profiling (LWP)说明。

- Fix the offset for `%arg` in 64−bit mode.

在 64 位模式下修复% arg 的偏移量。

- An undefined local macro (`%$`) no longer matches a global macro with the same name.

未定义的本地宏(% $)不再匹配同名的全局宏。

- Fix NULL dereference on too long local labels.

修正对过长的本地标签的 NULL 取消引用。

C.1.46 Version 2.07

C. 1.46 Version 2.07

- NASM is now under the 2−clause BSD license. See section 1.1.1.

NASM 现在在 BSD 许可证的 2 条款之下。参见第 1.1.1 节。

- Fix the section type for the `.strtab` section in the `elf64` output format.

在 elf64 输出格式中修正.strtab 部分的节类型。

- Fix the handling of `COMMON` directives in the `obj` output format.

以 obj 输出格式修正 COMMON 指令的处理。

- New `ith` and `srec` output formats; these are variants of the `bin` output format which output Intel hex and Motorola S−records, respectively. See section 7.2 and section 7.3.

新的 ith 和 srec 输出格式; 这些是分别输出 Intel 十六进制和 Motorola s 记录的 bin 输出格式的变体。参见第 7.2 节和第 7.3 节。

247

- `rdf2ihx` replaced with an enhanced `rdf2bin`, which can output binary, COM, Intel hex or Motorola S−records.

Rdf2ihx 被一个增强的 rdf2bin 代替，它可以输出二进制，COM，Intel 十六进制或摩托罗拉 s 记录。

- The Windows installer now puts the NASM directory first in the `PATH` of the "NASM Shell".

Windows installer 现在将 NASM 目录放在“ NASM Shell”的 PATH 中的第一位。

- Revert the early expansion behavior of `%+` to pre−2.06 behavior: `%+` is only expanded late.

恢复% + 的早期扩展行为到 2.06 之前的行为:% + 只是后期扩展。

- Yet another Mach−O alignment fix.

又一个马赫 -o 校准修正。

- Don’t delete the list file on errors. Also, include error and warning information in the list file.

不要删除关于错误的列表文件，同时在列表文件中包含错误和警告信息。

- Support for 64−bit Mach−O output, see section 7.8.

支持 64 位 Mach-o 输出，参见 7.8 节。

- Fix assert failure on certain operations that involve strings with high−bit bytes.

修复某些操作中涉及高位字节字符串的断言失败。

C.1.47 Version 2.06
C. 1.47 版本 2.06

- This release is dedicated to the memory of Charles A. Crayne, long time NASM developer as well as moderator of `comp.lang.asm.x86` and author of the book *Serious Assembler*. We miss you, Chuck.

这个版本是献给 Charles a。 Crayne 的，他是 NASM 的长期开发者，comp.lang.asm.x86 的主持人，也是 Serious Assembler 一书的作者。我们想念你，Chuck。

- Support for indirect macro expansion (`%[...]`). See section 4.1.3.

支持间接宏展开(% [...])参见第 4.1.3 节。

- `%pop` can now take an argument, see section 4.7.1.

% pop 现在可以接受一个参数，参见第 4.7.1 节。

- The argument to `%use` is no longer macro−expanded. Use `%[...]` if macro expansion is desired.

% Use 的参数不再是宏展开的。如果需要宏展开，请使用% [...]。

- Support for thread−local storage in ELF32 and ELF64. See section 7.9.4.

在 elf32 和 elf64 中支持线程本地存储。参见 7.9.4 节。

- Fix crash on `%ifmacro` without an argument.

修正了不带参数的% ifmacro 崩溃问题。

- Correct the arguments to the `POPCNT` instruction.

将参数更正为 POPCNT 指令。

- Fix section alignment in the Mach−O format.

以 Mach-o 格式修正节对齐方式。

- Update AVX support to version 5 of the Intel specification.

更新 AVX 支持到 Intel 规范的第 5 版。

- Fix the handling of accesses to context−local macros from higher levels in the context stack.

修复从上下文堆栈的更高级别对上下文本地宏的访问的处理。

- Treat `WAIT` as a prefix rather than as an instruction, thereby allowing constructs like `O16 FSAVE` to work correctly.

把 WAIT 当作一个前缀而不是一个指令，从而允许像 O16 FSAVE 这样的结构正常工作。

- Support for structures with a non−zero base offset. See section 4.11.11.

非零基偏移结构的支持参见第 4.11.11 节。

- Correctly handle preprocessor token concatenation (see section 4.3.9) involving floating−point numbers.

正确处理涉及浮点数的预处理令牌连接(见第 4.3.9 节)。

- The `PINSR` series of instructions have been corrected and rationalized.

PINSR 系列指令已经得到修正和合理化。

- Removed AMD SSE5, replaced with the new XOP/FMA4/CVT16 (rev 3.03) spec.

删除 AMD SSE5，取而代之的是新的 XOP/FMA4/CVT16(rev 3.03)规范。

- The ELF backends no longer automatically generate a `.comment` section.

ELF 后端不再自动生成.comment 部分。

- Add additional "well−known" ELF sections with default attributes. See section 7.9.2.

添加额外的带有默认属性的“众所周知的”ELF 部分。参见第 7.9.2 节。

C.1.48 Version 2.05.01
C. 1.48 版本 2.05.01

- Fix the `-w`/`-W` option parsing, which was broken in NASM 2.05.

修复 -w/-w 选项解析，它在 NASM 2.05 中被破坏。

C.1.49 Version 2.05
C. 1.49 版本 2.05

- Fix redundant REX.W prefix on `JMP reg64`.

修复 JMP reg64 上冗余的 REX.W 前缀。

- Make the behaviour of `-O0` match NASM 0.98 legacy behavior. See section 2.1.23.

使 -o0 的行为与 nasm0.98 的遗留行为匹配参见 2.1.23 节。

- `-w-user` can be used to suppress the output of `%warning` directives. See section 2.1.25.

用户可以用来抑制`%` 警告指令的输出。参见第 `2.1.25` 节。

- Fix bug where `ALIGN` would issue a full alignment datum instead of zero bytes.

修正 ALIGN 将发出完整对齐数据而不是零字节的错误。

- Fix offsets in list files.

修正列表文件中的偏移量。

- Fix `%include` inside multi−line macros or loops.

Fix% 包含在多行宏或循环中。

- Fix error where NASM would generate a spurious warning on valid optimizations of immediate values.

Fix 错误，其中 NASM 会对即时值的有效优化生成虚假警告。

- Fix arguments to a number of the `CVT` SSE instructions.

Fix 一些 CVT SSE 指令的参数。

- Fix RIP−relative offsets when the instruction carries an immediate.

修正 RIP-当指令带有一个直接的。

- Massive overhaul of the ELF64 backend for spec compliance.

对 elf64 后端进行大规模检修，以符合规范要求。

- Fix the Geode `PFRCPV` and `PFRSQRTV` instruction.

修复 Geode PFRCPV 和 PFRSQRTV 指令。

- Fix the SSE 4.2 `CRC32` instruction.

修复 SSE 4.2 crc32 指令。

C.1.50 Version 2.04
C. 1.50 版本 2.04

- Sanitize macro handing in the `%error` directive.

Sanitize 宏处理中的% error 指令。

- New `%warning` directive to issue user−controlled warnings.

发出用户控制的警告的新% 警告指令。

- `%error` directives are now deferred to the final assembly phase.

% 错误指令现在被推迟到最后的组装阶段。

- New `%fatal` directive to immediately terminate assembly.

命令立即终止程序集。

- New `%strcat` directive to join quoted strings together.

New% strcat 指令将引用的字符串连接在一起。

- New `%use` macro directive to support standard macro directives. See section 4.6.4.

New% 使用宏指令来支持标准宏指令。参见第 4.6.4 节。

- Excess default parameters to `%macro` now issues a warning by default. See section 4.3.

默认情况下,% macro 的过多默认参数现在会发出警告。参见第 4.3 节。

- Fix `%ifn` and `%elifn`.

修复% ifn 和% elifn。

- Fix nested `%else` clauses.

Fix 嵌套的% else 子句。

- Correct the handling of nested `%rep`s.

纠正嵌套% rep 的处理。

- New `%unmacro` directive to undeclare a multi−line macro. See section 4.3.12.

New% unmacro 指令取消对多行宏的声明参见第 4.3.12 节。

- Builtin macro `__PASS__` which expands to the current assembly pass. See section 4.11.10.

内建宏 _ _ PASS _ _ 扩展到当前的程序集通道. 参见第 4.11.10 节。

- `__utf16__` and `__utf32__` operators to generate UTF−16 and UTF−32 strings. See section 3.4.5.

_ _ utf16 _ _ 和 _ _ utf32 _ _ 运算符来生成 UTF-16 和 UTF-32 字符串。

- Fix bug in case−insensitive matching when compiled on platforms that don’t use the `configure` script. Of the official release binaries, that only affected the OS/2 binary.

在不使用配置脚本的平台上编译时，修复了不区分大小写匹配的错误。对于官方发布的二进制文件，这只会影响 OS/2 二进制文件。

- Support for x87 packed BCD constants. See section 3.4.7.

支持 x87 包装的 BCD 常量，参见 3.4.7 节。

- Correct the `LTR` and `SLDT` instructions in 64−bit mode.

在 64 位模式下修正 LTR 和 SLDT 指令。

- Fix unnecessary REX.W prefix on indirect jumps in 64−bit mode.

修正 64 位模式下间接跳转时不必要的 REX.W 前缀。

- Add AVX versions of the AES instructions (`VAES`...).

添加 AES 指令的 AVX 版本(VAES...)。

- Fix the 256−bit FMA instructions.

修复 256 位 FMA 指令。

- Add 256−bit AVX stores per the latest AVX spec.

根据最新的 AVX 规范添加 256 位 AVX 商店。

- VIA XCRYPT instructions can now be written either with or without `REP`, apparently different versions of the VIA spec wrote them differently.

VIA XCRYPT 的指令现在可以使用或不使用 REP 编写，显然 VIA 规范的不同版本编写了不同的指令。

- Add missing 64−bit `MOVNTI` instruction.

添加缺少的 64 位 MOVNTI 指令。

- Fix the operand size of `VMREAD` and `VMWRITE`.

修正 VMREAD 和 VMWRITE 的操作数大小。

- Numerous bug fixes, especially to the AES, AVX and VTX instructions.

许多错误修复，特别是 AES，AVX 和 VTX 指令。

- The optimizer now always runs until it converges. It also runs even when disabled, but doesn't optimize. This allows most forward references to be resolved properly.

优化器现在总是运行到收敛为止。它甚至在禁用的情况下也会运行，但不会进行优化。这样就可以正确地解析大多数前向引用。

- `%push` no longer needs a context identifier; omitting the context identifier results in an anonymous context.

`% push` 不再需要上下文标识符；省略上下文标识符会导致匿名上下文。

C.1.51 Version 2.03.01
C. 1.51 版本 2.03.01

- Fix buffer overflow in the listing module.

修复清单模块中的缓冲区溢出。

- Fix the handling of hexadecimal escape codes in '...' strings.

修正"..."字符串中十六进制转义码的处理。

- The Postscript/PDF documentation has been reformatted.

Postscript/PDF 文档已被重新格式化。

- The `-F` option now implies `-g`.

F 选项现在意味着 -g。

C.1.52 Version 2.03
C. 1.52 Version 2.03 c. 1.52 Version 2.03

- Add support for Intel AVX, CLMUL and FMA instructions, including YMM registers.

添加对 Intel AVX、 CLMUL 和 FMA 指令的支持，包括 YMM 寄存器。

- `dy`, `resy` and `yword` for 32−byte operands.

`Dy`，`resy` 和 `yword` 用于 32 字节的操作数。

- Fix some SSE5 instructions.

修正一些 sse5 指令。

- Intel `INVEPT`, `INVVPID` and `MOVBE` instructions.

Intel INVEPT，INVVPID 和 MOVBE 指令。

- Fix checking for critical expressions when the optimizer is enabled.

Fix 在启用优化器时检查关键表达式。

- Support the DWARF debugging format for ELF targets.

支持 ELF 目标的 DWARF 调试格式。

- Fix optimizations of signed bytes.

修正有符号字节的优化。

- Fix operation on bigendian machines.

二进制机器上的 Fix 操作。

- Fix buffer overflow in the preprocessor.

修正预处理器中的缓冲区溢出。

- `SAFESEH` support for Win32, `IMAGEREL` for Win64 (SEH).

支持 `win32` 的 `SAFESEH`，支持 `win64` 的 `IMAGEREL (SEH)`。

- `%?` and `%??` to refer to the name of a macro itself. In particular, `%idefine keyword $%?` can be used to make a keyword "disappear".

`%?` 还有`%` ？指的是宏本身的名称。特别是，`%` 定义关键字 `$%` ？可以用来使关键字"消失"。

- New options for dependency generation: `-MD`, `-MF`, `-MP`, `-MT`, `-MQ`.

依赖生成的新选项:-MD,-MF,-MP,-MT,-MQ。

- New preprocessor directives `%pathsearch` and `%depend`; INCBIN reimplemented as a macro.

新的预处理器指令% path search 和% depend; INCBIN 重新实现为宏。

- `%include` now resolves macros in a sane manner.

`% include` 现在以理智的方式解析宏。

- `%substr` can now be used to get other than one−character substrings.

`% substr` 现在可以用来获取除单字符子字符串之外的其他字符。

- New type of character/string constants, using backquotes (`` `...` ``), which support C−style escape sequences.

新类型的字符/字符串常量，使用后引号(‘ ...’)，支持 c 风格的转义序列。

- `%defstr` and `%idefstr` to stringize macro definitions before creation.

`% defstr` 和`% idefstr` 在创建之前对宏定义进行字符串化。

- Fix forward references used in `EQU` statements.

修正 EQU 语句中使用的正向引用。

C.1.53 Version 2.02
C. 1.53 版本 2.02

- Additional fixes for MMX operands with explicit `qword`, as well as (hopefully) SSE operands with `oword`.

MMX 操作数使用显式 qword 以及(希望如此) SSE 操作数使用 oword 的附加修正。

- Fix handling of truncated strings with `DO`.

用 DO 修复对截断字符串的处理。

- Fix segfaults due to memory overwrites when floating−point constants were used.

Fix 使用浮点常量时由于内存覆盖引起的 segfault。

- Fix segfaults due to missing include files.

修正由于缺少包含文件而产生的错误。

- Fix OpenWatcom Makefiles for DOS and OS/2.

修复 OpenWatcom Makefiles for DOS and OS/2。

- Add autogenerated instruction list back into the documentation.

在文档中添加自动生成的指令表。

- ELF: Fix segfault when generating stabs, and no symbols have been defined.

当生成刺时，修正了错误，并且没有定义符号。

- ELF: Experimental support for DWARF debugging information.

ELF: 对 DWARF 调试信息的实验性支持。

- New compile date and time standard macros.

新的编译日期和时间标准宏。

- `%ifnum` now returns true for negative numbers.

`% ifnum` 现在对负数返回 `true`。

- New `%iftoken` test for a single token.

对单个令牌进行 New% iftoken 测试。

- New `%ifempty` test for empty expansion.

New% ifempty test for empty expansion.

- Add support for the `XSAVE` instruction group.

添加对 XSAVE 指令组的支持。

- Makefile for Netware/gcc.

Makefile for Netware/gcc.

- Fix issue with some warnings getting emitted way too many times.

修正了某些警告发出太多次的问题。

- Autogenerated instruction list added to the documentation.

自动生成的指令表添加到文档中。

C.1.54 Version 2.01
C. 1.54 版本 2.01

- Fix the handling of MMX registers with explicit `qword` tags on memory (broken in 2.00 due to 64−bit changes.)

修正了在内存中使用显式 qword 标签处理 MMX 寄存器的问题(由于 64 位的更改，2.00 中断)

- Fix the PREFETCH instructions.

修正 PREFETCH 指令。

- Fix the documentation.

修复文档。

- Fix debugging info when using `-f elf` (backwards compatibility alias for `-f elf32`).

•修正使用 -f elf 时的调试信息(-f elf32 的向后兼容别名)。

- Man pages for rdoff tools (from the Debian project.)

Rdoff 工具的 Man 页面(来自 Debian 项目)

- ELF: handle large numbers of sections.

处理大量的部分。

- Fix corrupt output when the optimizer runs out of passes.

修复优化器用完通道时的损坏输出。

C.1.55 Version 2.00
C. 1.55 版本 2.00

- Added c99 data−type compliance.

增加了 c99 数据类型兼容性。

- Added general x86−64 support.

增加了通用 x86-64 支持。

- Added win64 (x86−64 COFF) output format.
增加了 win64(x86-64 COFF)输出格式。

- Added `__BITS__` standard macro.
添加了标准宏。

- Renamed the `elf` output format to `elf32` for clarity.
为了清晰起见，将 elf 输出格式重命名为 elf32。

- Added `elf64` and `macho` (MacOS X) output formats.
添加了 elf64 和 macho (macosx)输出格式。

- Added Numeric constants in `dq` directive.
在 dq 指令中添加了 Numeric 常量。

- Added `oword`, `do` and `reso` pseudo operands.
增加了一个单词，do 和 reso 伪操作数。

- Allow underscores in numbers.
允许数字下划线。

- Added 8−, 16− and 128−bit floating−point formats.
添加了 8 位、16 位和 128 位浮点格式。

- Added binary, octal and hexadecimal floating−point.
增加二进制，八进制和十六进制浮点数。

- Correct the generation of floating−point constants.
更正浮点常数的生成。

- Added floating−point option control.
添加浮点选项控制。

- Added Infinity and NaN floating point support.
增加了对 Infinity 和 NaN 浮点的支持。

- Added ELF Symbol Visibility support.
增加了 ELF 符号可见性支持。

251

- Added setting OSABI value in ELF header directive.
在 ELF 头部指令中添加了设置 OSABI 值。

- Added Generate Makefile Dependencies option.
添加生成 Makefile 依赖项选项。

- Added Unlimited Optimization Passes option.
无限优化通过选项。

- Added `%IFN` and `%ELIFN` support.
添加% IFN 和% ELIFN 支持。

- Added Logical Negation Operator.
增加了逻辑非运算符。

- Enhanced Stack Relative Preprocessor Directives.
增强的堆栈相对预处理器指令。

- Enhanced ELF Debug Formats.
增强的 ELF 调试格式。

- Enhanced Send Errors to a File option.
增强的"向文件发送错误"选项。

- Added SSSE3, SSE4.1, SSE4.2, SSE5 support.
增加了 SSSE3，SSE4.1，SSE4.2，sse5 支持。

- Added a large number of additional instructions.
添加了大量的附加指令。

- Significant performance improvements.
显著的性能改进。

- `-w+warning` and `-w-warning` can now be written as –Wwarning and –Wno−warning, respectively. See section 2.1.25.
- `w + warning` 和`-w-warning` 现在可以分别写为`-Wwarning` 和`-Wno-warning`。参见第 `2.1.25` 节。

- Add `-w+error` to treat warnings as errors. See section 2.1.25.
将警告视为错误的 Add-w + error 参见第 2.1.25 节。

- Add `-w+all` and `-w-all` to enable or disable all suppressible warnings. See section 2.1.25.
Add-w + all 和-w-all 启用或禁用所有可抑制的警告。参见第 2.1.25 节。

C.2 NASM 0.98 Series
C. 2 NASM 0.98 系列

The 0.98 series was the production versions of NASM from 1999 to 2007.
0.98 系列是 NASM 从 1999 年到 2007 年的生产版本。

C.2.1 Version 0.98.39
C. 2.1 版本 0.98.39

- fix buffer overflow
修复缓冲区溢出

- fix outas86's `.bss` handling
修复 86 的 bss 处理

- "make spotless" no longer deletes config.h.in.
" make spotless"不再删除 config.h.in。

- `%(el)if(n)idn` insensitivity to string quotes difference (#809300).
如果`(n) idn` 对字符串引号不敏感`(# 809300)`。

- (nasm.c)`__OUTPUT_FORMAT__` changed to string value instead of symbol.

(nasm.c) _ _ OUTPUT _ format _ _ 更改为字符串值而不是符号。

C.2.2 Version 0.98.38
C. 2.2 版本 0.98.38

- Add Makefile for 16−bit DOS binaries under OpenWatcom, and modify `mkdep.pl` to be able to generate completely pathless dependencies, as required by OpenWatcom wmake (it supports path searches, but not explicit paths.)

在 OpenWatcom 下为 16 位 DOS 二进制文件添加 Makefile，并修改 mkdep.pl，以便能够根据 OpenWatcom wmake 的要求生成完全无路径的依赖项(它支持路径搜索，但不支持显式路径)

- Fix the `STR` instruction.

修复 STR 指令。

- Fix the ELF output format, which was broken under certain circumstances due to the addition of stabs support.

修正了 ELF 输出格式，在某些情况下由于添加了 stab 支持而被破坏。

- Quick−fix Borland format debug−info for `−f obj`

Quick-fix Borland 格式 debug-info for-f obj

- Fix for `%rep` with no arguments (#560568)

修复无参数的% rep (# 560568)

- Fix concatenation of preprocessor function call (#794686)

Fix 预处理器函数调用的连接(# 794686)

- Fix long label causes coredump (#677841)

修正长标签导致内核转储的问题(# 677841)

- Use autoheader as well as autoconf to keep configure from generating ridiculously long command lines.

使用 autoheader 和 autoconf 来防止 configure 生成荒谬的长命令行。

- Make sure that all of the formats which support debugging output actually will suppress debugging output when `-g` not specified.

确保所有支持调试输出的格式在没有指定 -g 时会抑制调试输出。

C.2.3 Version 0.98.37
C. 2.3 版本 0.98.37

- Paths given in `-I` switch searched for `incbin`–ed as well as `%include`–ed files.

在-i 开关中给出的路径搜索了 inbin-ed 以及% include-ed 文件。

- Added stabs debugging for the ELF output format, patch from Martin Wawro.

添加了针对 ELF 输出格式的 stab 调试，来自 Martin Wawro 的补丁。

- Fix `output/outbin.c` to allow origin > 80000000h.

Fix output/outbin.c 允许 origin > 8000000h。

- Make `-U` switch work.

让 u 开关工作。

- Fix the use of relative offsets with explicit prefixes, e.g. `a32 loop foo`.

修正使用带有显式前缀的相对偏移量，例如 a32 循环 foo。

- Remove `backslash()`.

删除反斜杠()。

- Fix the `SMSW` and `SLDT` instructions.

修复 SMSW 和 SLDT 指令。

- `-O2` and `-O3` are no longer aliases for `-O10` and `-O15`. If you mean the latter, please say so! :)

- o2 和-o3 不再是-o10 和-o15 的别名。如果你指的是后者，请说出来！:)

C.2.4 Version 0.98.36
C. 2.4 版本 0.98.36

- Update rdoff – librarian/archiver – common rec – docs!

更新 rdoff-library/archiver-common rec-docs！

- Fix signed/unsigned problems.

修复已签名/未签名的问题。

- Fix `JMP FAR label` and `CALL FAR label`.

修正 JMP FAR 标签和 CALL FAR 标签。

- Add new multisection support – map files – fix align bug

添加新的多节支持-地图文件-修复对齐错误

- Fix sysexit, movhps/movlps reg,reg bugs in insns.dat

修正了 insns.dat 中的 sysexit、 movhps/movlps reg、 reg bug

- `Q` or `O` suffixes indicate octal

Q 或 o 后缀表示八进制

- Support Prescott new instructions (PNI).

支持 Prescott 新指令(PNI)。

- Cyrix `XSTORE` instruction.

Cyrix XSTORE 指令。

C.2.5 Version 0.98.35
C. 2.5 版本 0.98.35

- Fix build failure on 16−bit DOS (Makefile.bc3 workaround for compiler bug.)

修复 16 位 DOS 上的构建失败(makefile.bc3 编译器错误解决方案)

- Fix dependencies and compiler warnings.

修复依赖项和编译器警告。

- Add "const" in a number of places.

在许多地方添加“ const”。

- Add –X option to specify error reporting format (use –Xvc to integrate with Microsoft Visual Studio.)

Add-x 选项指定错误报告格式(使用 -Xvc 与 microsoftvisualstudio 集成)

- Minor changes for code legibility.

代码易读性的小改动。

- Drop use of tmpnam() in rdoff (security fix.)

放弃在 rdoff (安全修复)中使用 tmpnam ()

C.2.6 Version 0.98.34
C. 2.6 版本 0.98.34

- Correct additional address−size vs. operand−size confusions.

纠正附加地址大小与操作数大小的混淆。

- Generate dependencies for all Makefiles automatically.

自动生成所有 makefile 的依赖项。

- Add support for unimplemented (but theoretically available) registers such as tr0 and cr5. Segment registers 6 and 7 are called segr6 and segr7 for the operations which they can be represented.

Add 支持未实现(但理论上可用)的寄存器，如 tr0 和 cr5。段寄存器 6 和 7 被称为 segr6 和 segr7，用于它们可以表示的操作。

- Correct some disassembler bugs related to redundant address−size prefixes. Some work still remains in this area.

纠正一些与冗余地址大小前缀相关的反汇编程序错误，这个领域还有一些工作要做。

- Correctly generate an error for things like "SEG eax".

正确地为“ SEG eax”之类的东西生成一个错误。

- Add the JMPE instruction, enabled by "CPU IA64".

添加由“ CPU IA64”启用的 JMPE 指令。

253

- Correct compilation on newer gcc/glibc platforms.

更新的 gcc/glibc 平台上的正确编译。

- Issue an error on things like "jmp far eax".

在诸如“ jmpfar eax”之类的事情上发出错误。

C.2.7 Version 0.98.33
C. 2.7 版本 0.98.33

- New __NASM_PATCHLEVEL__ and __NASM_VERSION_ID__ standard macros to round out the version−query macros. version.pl now understands X.YYplWW or X.YY.ZZplWW as a version number, equivalent to X.YY.ZZ.WW (or X.YY.0.WW, as appropriate).

新的 _ NASM _ patchlevel _ _ 和 _ NASM _ version _ id _ _ 标准宏来完善版本查询宏。Pl 现在将 X.YYplWW 或 X.YY.ZZplWW 理解为一个版本号，相当于 X.YY.ZZ.WW (或 X.YY. 0。WW，酌情)。

- New keyword "strict" to disable the optimization of specific operands.

New 关键字“ strict”禁用特定操作数的优化。

- Fix the handing of size overrides with JMP instructions (instructions such as "jmp dword foo".)

使用 JMP 指令(如“ jmpdwordfoo”等指令)修复大小重写的处理

- Fix the handling of "ABSOLUTE label", where "label" points into a relocatable segment.

修正了“ ABSOLUTE label”的处理，“ label”指向一个可重定位段。

- Fix OBJ output format with lots of externs.

修正 OBJ 输出格式与大量的外部程序。

- More documentation updates.

更多文档更新。

- Add –Ov option to get verbose information about optimizations.

Add-Ov 选项以获取有关优化的详细信息。

- Undo a braindead change which broke `%elif` directives.

撤销破坏% elif 指令的脑死亡更改。

- Makefile updates.

Makefile 更新。

C.2.8 Version 0.98.32
C. 2.8 版本 0.98.32

- Fix NASM crashing when `%macro` directives were left unterminated.

修正当% 宏指令未被终止时 NASM 崩溃的问题。

- Lots of documentation updates.

大量的文档更新。

- Complete rewrite of the PostScript/PDF documentation generator.

完全重写 PostScript/PDF 文件产生器。

- The MS Visual C++ Makefile was updated and corrected.

MS Visual c + + Makefile 被更新和修正。

- Recognize .rodata as a standard section name in ELF.

将 rodata 识别为 ELF 中的标准节名称。

- Fix some obsolete Perl4−isms in Perl scripts.

修复 Perl 脚本中一些过时的 Perl4-isms。

- Fix configure.in to work with autoconf 2.5x.

Fix configure.in 以使用 autoconf2.5 x。

- Fix a couple of "make cleaner" misses.

修正了一些“ make cleaner”的错误。

- Make the normal "./configure && make" work with Cygwin.

使用 Cygwin 进行正常的“ ./configure & & Make”操作。

C.2.9 Version 0.98.31
C. 2.9 版本 0.98.31

- Correctly build in a separate object directory again.

再次正确地在一个单独的对象目录中构建。

- Derive all references to the version number from the version file.

从版本文件中获取对版本号的所有引用。

- New standard macros __NASM_SUBMINOR__ and __NASM_VER__ macros.

新的标准宏 _ NASM _ subminor _ 和 _ NASM _ ver _ 宏。

- Lots of Makefile updates and bug fixes.

大量的 Makefile 更新和 bug 修复。

- New `%ifmacro` directive to test for multiline macros.

测试多行宏的新% ifmacro 指令。

- Documentation updates.

文档更新。

- Fixes for 16−bit OBJ format output.

修复 16 位 OBJ 格式输出。

- Changed the NASM environment variable to NASMENV.

将 NASM 环境变量更改为 NASMENV。

C.2.10 Version 0.98.30
C. 2.10 版本 0.98.30

- Changed doc files a lot: completely removed old READMExx and Wishlist files, incorporating all information in CHANGES and TODO.

大量更改文档文件: 完全删除旧的 READMExx 和 Wishlist 文件，整合了 CHANGES 和 TODO 中的所有信息。

- I waited a long time to rename zoutieee.c to (original) outieee.c
我等了很久才把 zoutieee.c 改名为 outieee.c

- moved all output modules to output/ subdirectory.
将所有输出模块移至输出/子目录。

- Added ’make strip’ target to strip debug info from nasm & ndisasm.
添加“ make strip”目标，从 nasm & ndisasm 中剥离调试信息。

- Added INSTALL file with installation instructions.
添加了具有安装说明的 INSTALL 文件。

- Added –v option description to nasm man.
为 nasm man 添加了 -v 选项描述。

- Added dist makefile target to produce source distributions.
添加 dist makefile 目标以生成源代码发行版。

- 16−bit support for ELF output format (GNU extension, but useful.)
16 位支持 ELF 输出格式(GNU 扩展，但很有用)

C.2.11 Version 0.98.28
C. 2.11 版本 0.98.28

- Fastcooked this for Debian’s Woody release: Frank applied the INCBIN bug patch to 0.98.25alt and called it 0.98.28 to not confuse poor little apt−get.
Fastcook 为 Debian 的 Woody 版本做了这个: Frank 将 INCBIN bug 补丁应用到 0.98.25 alt，并将其命名为 0.98.28，以免混淆可怜的小 apt-get。

C.2.12 Version 0.98.26
C. 2.12 版本 0.98.26

- Reorganised files even better from 0.98.25alt
•从 0.98.25 alt 更好地重新组织文件

C.2.13 Version 0.98.25alt
C. 2.13 版本 0.98.25 alt

- Prettified the source tree. Moved files to more reasonable places.
对源代码树进行了美化，将文件移动到更合理的位置。

- Added findleak.pl script to misc/ directory.
将 findleak.pl 脚本添加到 misc/目录。

- Attempted to fix doc.
尝试修复 doc。

C.2.14 Version 0.98.25
C. 2.14 版本 0.98.25

- Line continuation character \.
行继续字符。

- Docs inadvertantly reverted – "dos packaging".
Docs 无意中恢复了-“ dos 包装”。

C.2.15 Version 0.98.24p1
C. 2.15 Version 0.98.24 p1

- FIXME: Someone, document this please.
• FIXME: 请记录下来。

C.2.16 Version 0.98.24
C. 2.16 版本 0.98.24

- Documentation – Ndisasm doc added to Nasm.doc.

* 文件-Ndisasm 文件添加到 Nasm.doc。

C.2.17 Version 0.98.23
C. 2.17 版本 0.98.23

- Attempted to remove rdoff version1

试图删除 rdoff 版本 1

- Lino Mastrodomenico's patches to preproc.c (%$$ bug?).

Lino Mastrodomenico 对 preproc.c 的补丁(% $$bug。

C.2.18 Version 0.98.22
C. 2.18 版本 0.98.22

- Update rdoff2 – attempt to remove v1.

更新 rdoff2-尝试删除 v1。

C.2.19 Version 0.98.21
C. 2.19 版本 0.98.21

- Optimization fixes.

优化修复。

C.2.20 Version 0.98.20
C. 2.20 版本 0.98.20

- Optimization fixes.

优化修复。

C.2.21 Version 0.98.19
C. 2.21 版本 0.98.19

- H. J. Lu's patch back out.

陆兆禧(H.j。 Lu)的补丁脱落。

C.2.22 Version 0.98.18
C. 2.22 版本 0.98.18

- Added ".rdata" to "−f win32".
- 在「-f win32」中加入「 . rdata 」。

C.2.23 Version 0.98.17
C. 2.23 版本 0.98.17

- H. J. Lu’s "bogus elf" patch. (Red Hat problem?)
- H.j. Lu 的“冒牌精灵”补丁(红帽子问题?)

C.2.24 Version 0.98.16
C. 2.24 Version 0.98.16 c. 2.240.98.16

- Fix whitespace before "[section ..." bug.

•修正“[section...”bug 之前的空格。

C.2.25 Version 0.98.15
C. 2.25 版本 0.98.15

- Rdoff changes (?).

  Rdoff 更改(?)。

- Fix fixes to memory leaks.

  修复内存泄漏。

C.2.26 Version 0.98.14
C. 2.26 版本 0.98.14

- Fix memory leaks.

  修复内存泄漏。

C.2.27 Version 0.98.13
C. 2.27 版本 0.98.13

- There was no 0.98.13

•没有 0.98.13

C.2.28 Version 0.98.12
C. 2.28 版本 0.98.12

- Update optimization (new function of "−O1")

  更新优化(“-O1”的新功能)

- Changes to test/bintest.asm (?).

  Test/bintest.asm (?)的更改。

C.2.29 Version 0.98.11
C. 2.29 版本 0.98.11

- Optimization changes.

  优化变化。

- Ndisasm fixed.

  Ndisasm 修复。

C.2.30 Version 0.98.10
C. 2.30 版本 0.98.10

- There was no 0.98.10

•没有 0.98.10

C.2.31 Version 0.98.09
C. 2.31 版 0.98.09

- Add multiple sections support to "−f bin".
为“-f bin”添加多节支持。

- Changed GLOBAL_TEMP_BASE in outelf.c from 6 to 15.
将 outelf.c 中的 GLOBAL _ temp _ base 从 6 更改为 15。

- Add "−v" as an alias to the "−r" switch.
添加“-v”作为“-r”开关的别名。

- Remove "#ifdef" from Tasm compatibility options.
从 Tasm 兼容性选项中删除“ # ifdef”。

- Remove redundant size−overrides on "mov ds, ex", etc.
删除冗余大小-重写“ mov ds，ex”等。

- Fixes to SSE2, other insns.dat (?).
修复了 sse2 和其他 insns.dat (?)。

- Enable uppercase "I" and "P" switches.
启用大写的“ i”和“ p”开关。

- Case insinsitive "seg" and "wrt".
不区分大小写的“ seg”和“ wrt”。

- Update install.sh (?).
更新 install.sh (。

- Allocate tokens in blocks.
以块为单位分配令牌。

- Improve "invalid effective address" messages.
改进“无效的有效地址”消息。

C.2.32 Version 0.98.08
C. 2.32 版本 0.98.08

• Add "`%strlen`" and "`%substr`" macro operators
添加“% strlen”和“% substr”宏操作符

• Fixed broken c16.mac.
修正了 c16.mac 中断的问题。

• Unterminated string error reported.
报告未终止的字符串错误。

• Fixed bugs as per 0.98bf
根据 0.98 bf 修正错误

C.2.33 Version 0.98.09b with John Coffman patches released 28−Oct−2001
C. 2.33 Version 0.98.09 b with John Coffman patches released 28-Oct-2001 c. 2.33 Version 0.98.09 b with John Coffman patches 发布于 2001 年 10 月 28 日

Changes from 0.98.07 release to 98.09b as of 28−Oct−2001
截至 2001 年 10 月 28 日，从 0.98.07 版本更改为 98.09 b 版本

• More closely compatible with 0.98 when –O0 is implied or specified. Not strictly identical, since backward branches in range of short offsets are recognized, and signed byte values with no explicit size specification will be assembled as a single byte.
当隐含或指定 -o0 时，更接近于 0.98。不完全相同，因为可以识别短偏移量范围内的向后分支，并且没有明确大小规范的有符号字节值将被组装为单个字节。

• More forgiving with the PUSH instruction. 0.98 requires a size to be specified always. 0.98.09b will imply the size from the current BITS setting (16 or 32).
更宽容的 PUSH 指令。0.98 要求总是指定一个尺寸。0.98.09 b 意味着当前 BITS 设置(16 或 32)的大小。

• Changed definition of the optimization flag:
更改了优化标志的定义:

```
-O0     strict two-pass assembly, JMP and Jcc are
O0 严格的两通程序集，JMP 和 Jcc 是
        handled more like 0.98, except that
        back- ward JMPs are short, if possible.
        更接近于 0.98，但是如果可能的话，向后的 jmp 是
        短的。

-O1     strict two-pass assembly, but forward
- o1 严格的两次通过装配，但是向前
        branches are assembled with code
        guaranteed to reach; may produce larger
        code than -O0, but will produce successful
        assembly more often if branch offset sizes
        are not specified.
        分支装配的代码保证到达；可能会产生比 -o0 更大的代
        码，但如果没有指定分支偏移量大小，则会更经常地产生
        成功的装配。

-O2     multi-pass optimization, minimize branch
O2 多路优化，最小化分支
        offsets; also will minimize signed immed-
        iate bytes, overriding size specification.
        偏移量；也将最小化有符号的嵌入式字节，覆盖大小规范。
```

```
-O3     like -O2, but more passes taken, if needed
- o3 和 -o2 一样，但是如果需要的话需要更多的通过
```

C.2.34 Version 0.98.07 released 01/28/01
C. 2.34 版本 0.98.0701/28/01 发布

- Added Stepane Denis' SSE2 instructions to a *working* version of the code – some earlier versions were based on broken code – sorry 'bout that. version "0.98.07"

添加 Stepane Denis 的 sse2 指令到一个"工作"版本的代码——一些早期的版本是基于破碎的代码——抱歉。版本"0.98.07"

- Cosmetic modifications to nasm.c, nasm.h, AUTHORS, MODIFIED

对 nasm.c，nasm.h，AUTHORS，MODIFIED

C.2.35 Version 0.98.06f released 01/18/01
C. 2.35 版本 0.98.06 f 发布 01/18/01

- Add "metalbrain"s jecxz bug fix in insns.dat

在 insns.dat 中添加" metalbrain"的 jecxz bug 修复

- Alter nasmdoc.src to match – version "0.98.06f"

修改 nasmdoc.src 以匹配版本"0.98.06 f"

C.2.36 Version 0.98.06e released 01/09/01
C. 2.36 版本 0.98.06 e 版本 01/09/01

- Removed the "outforms.h" file – it appears to be someone's old backup of "outform.h". version "0.98.06e"

删除了" outforms.h"文件——它似乎是某人" outform.h"的旧备份。版本"0.98.06 e"

- fbk – finally added the fix for the "multiple %includes bug", known since 7/27/99 – reported originally (?) and sent to us by Austin Lunnen – he reports that John Fine had a fix within the day. Here it is...

Fbk-最终为" multiple% includes bug"添加了修复，这个 bug 自 99 年 7 月 27 日以来就已知——最初报道(?)并由 Austin Lunnen 发送给我们——他报告说 John Fine 当天就有了一个修复。就是这个。

- Nelson Rush resigns from the group. Big thanks to Nelson for his leadership and enthusiasm in getting these changes incorporated into Nasm!

纳尔逊 · 拉什从组织中辞职。非常感谢尼尔森的领导和热情，他将这些变化纳入了 Nasm！

- fbk – [list +], [list –] directives – ineptly implemented, should be re−written or removed, perhaps.

Fbk-[list +]，[list-]指令-实施不当，也许应该重写或删除。

- Brian Raiter / fbk – "elfso bug" fix – applied to aoutb format as well – testing might be desirable...

Brian Raiter/fbk-“ elfso bug”修复-应用于 aoutb 格式-测试可能是可取的..。

- James Seter – –postfix, –prefix command line switches.

James Seter-后缀,-前缀命令行开关。

- Yuri Zaporozhets – rdoff utility changes.

Yuri Zaporozhets-rdoff 实用程序改变。

C.2.37 Version 0.98p1
C. 2.37 Version 0.98 p1

- GAS−like palign (Panos Minos)

类气体排列(Panos Minos)

- FIXME: Someone, fill this in with details

FIXME: 来个人，把这个填上细节

C.2.38 Version 0.98bf (bug−fixed)
C. 2.38 Version 0.98 bf (bug-fixed)

- Fixed – elf and aoutb bug – shared libraries – multiple "%include" bug in "−f obj" – jcxz, jecxz bug – unrecognized option bug in ndisasm

Fixed-elf 和 aoutb bug-shared libraries-multiple“% include”bug in“-f obj”-jcxz，jecxz bug-unrecognized option bug in ndisasm

C.2.39 Version 0.98.03 with John Coffman’s changes released 27−Jul−2000
C. 2.39 Version 0.98.03 with John Coffman’s changes released 27-Jul-2000

- Added signed byte optimizations for the 0x81/0x83 class of instructions: ADC, ADD, AND, CMP, OR,

为 0x81/0x83 类指令添加了有符号字节优化: ADC，ADD，AND，CMP，OR,

SBB, SUB, XOR: when used as ’ADD reg16,imm’ or ’ADD reg32,imm.’ Also optimization of signed byte form of ’PUSH imm’ and ’IMUL reg,imm’/’IMUL reg,reg,imm.’ No size specification is needed.

SBB，SUB，XOR: 当用作‘ ADD reg16，imm’或‘ ADD reg32，imm’时还优化了’PUSH imm’和’IMUL reg，imm’/’IMUL reg，reg，imm’的有符号字节形式不需要规格说明。

- Added multi−pass JMP and Jcc offset optimization. Offsets on forward references will preferentially use the short form, without the need to code a specific size (short or near) for the branch. Added instructions for ’Jcc label’ to use the form ’Jnotcc $+3/JMP label’, in cases where a short offset is out of bounds. If compiling for a 386 or higher CPU, then the 386 form of Jcc will be used instead.

增加了多通 JMP 和 Jcc 偏移量优化。正向引用上的偏移量将优先使用短格式，而不需要为分支编码特定的大小(短格式或近格式)。为“ Jcc 标签”添加了使用形式“ Jnotcc $+ 3/JMP 标签”的说明，以防止短偏移量超出范围。如果编译为 386 或更高的 CPU，那么将使用 Jcc 的 386 格式。

This feature is controlled by a new command−line switch: "O", (upper case letter O). "−O0" reverts the assembler to no extra optimization passes, "−O1" allows up to 5 extra passes, and "−O2"(default), allows up to 10 extra optimization passes.

该特性由一个新的命令行开关“ o”(大写字母 o)控制- O0”将汇编程序恢复为没有额外的优化通道，“-O1”最多允许 5 个额外通道，“-O2”(默认)最多允许 10 个额外的优化通道。

- Added a new directive: ’cpu XXX’, where XXX is any of: 8086, 186, 286, 386, 486, 586, pentium, 686, PPro, P2, P3 or Katmai. All are case insensitive. All instructions will be selected only if they apply to the selected cpu or lower. Corrected a couple of bugs in cpu−dependence in ’insns.dat’.

添加了一个新指令: “ cpu XXX”，其中 XXX 的值为: 8086,186,286,386,486,586，pentium，686，PPro，P2，p3 或 Katmai。所有这些都不区分大小写。所有指令只有在应用于所选 cpu 或更低的 cpu 时才会被选中。修正了 insns.dat 中 cpu 依赖方面的一些错误。

- Added to ’standard.mac’, the "use16" and "use32" forms of the "bits 16/32" directive. This is nothing new, just conforms to a lot of other assemblers. (minor)

在“ standard.mac”中添加了“ bit16/32”指令的“ use16”和“ use32”形式。这不是什么新鲜事，只是符合许多其他汇编程序。(小)

- Changed label allocation from 320/32 (10000 labels @ 200K+) to 32/37 (1000 labels); makes running under DOS much easier. Since additional label space is allocated dynamically, this should have no effect on large programs with lots of labels. The 37 is a prime, believed to be better for hashing. (minor)

将标签分配从 320/32(10000 个标签@200K +)改为 32/37(1000 个标签)，使得在 DOS 下运行更加容易。由于额外的标签空间是动态分配的，这对于有很多标签的大型程序应该没有影响。37 是一个质数，被认为更适合散列。(次要)

C.2.40 Version 0.98.03

C. 2.40 版本 0.98.03

"Integrated patchfile 0.98−0.98.01. I call this version 0.98.03 for historical reasons: 0.98.02 was trashed." —John Coffman <johninsd@san.rr.com>, 27−Jul−2000

集成补丁文件 0.98-0.98.01。由于历史原因，我称这个版本为 0.98.03:0.98.02 被丢弃了。”约翰 · 考夫曼 2000 年 7 月 27 日

- Kendall Bennett’s SciTech MGL changes

Kendall Bennett 的 SciTech MGL 发生了变化

- Note that you must define "TASM_COMPAT" at compile−time to get the Tasm Ideal Mode compatibility.

注意你必须在编译时定义“ TASM _ compat”来获得 TASM 理想模式的兼容性。

- All changes can be compiled in and out using the TASM_COMPAT macros, and when compiled without TASM_COMPAT defined we get the exact same binary as the unmodified 0.98 sources.

所有更改都可以使用 tasm_compat 宏进行编译，如果在没有定义 tasm_compat 的情况下进行编译，我们将得到与未修改的 0.98 源完全相同的二进制文件。

- standard.mac, macros.c: Added macros to ignore TASM directives before first include

Standard.mac，macros.c: 添加宏以忽略 TASM 指令

• nasm.h: Added extern declaration for tasm_compatible_mode
Nasm.h: 为 tasm _ compatible _ mode 添加了外部声明

• nasm.c: Added global variable tasm_compatible_mode
增加了全局变量 tasm _ compatible _ mode

• Added command line switch for TASM compatible mode (−t)
为 TASM 兼容模式(- t)添加了命令行开关

• Changed version command line to reflect when compiled with TASM additions
更改版本命令行，以反映在使用 TASM 添加进行编译时的情况

• Added response file processing to allow all arguments on a single line (response file is @resp rather than –@resp for NASM format).
添加响应文件处理，允许所有参数在一行(响应文件是@resp 而不是 -@resp 为 NASM 格式)。

• labels.c: Changes islocal() macro to support TASM style @@local labels.
C: Changes islocal ()宏支持 TASM style@@ local label。

• Added islocalchar() macro to support TASM style @@local labels.
添加了 islocalchar ()宏来支持 TASM 样式@@ local label。

• parser.c: Added support for TASM style memory references (ie: mov [DWORD eax],10 rather than the NASM style mov DWORD [eax],10).
C: 增加了对 TASM 样式内存引用的支持(即: mov [DWORD eax]，而不是 NASM 样式的 mov DWORD [eax]，10)。

• preproc.c: Added new directives, `%arg`, `%local`, `%stacksize` to directives table
Preproc.c: 添加了新的指令,% arg,% local,% stacksize 到指令表

• Added support for TASM style directives without a leading % symbol.
添加了对没有前导% 符号的 TASM 样式指令的支持。

• Integrated a block of changes from Andrew Zabolotny <bit@eltech.ru>:
整合了 Andrew Zabolotny 的一些变化:

• A new keyword `%xdefine` and its case−insensitive counterpart `%ixdefine`. They work almost the same way as `%define` and `%idefine` but expand the definition immediately, not on the invocation. Something like a cross between `%define` and `%assign`. The "x" suffix stands for "eXpand", so "xdefine" can be deciphered as "expand−and−define". Thus you can do things like this:
一个新的关键字% xdefine 及其不区分大小写的对应关键字% ixdefine。它们的工作方式与% define 和% idedefine 几乎相同，但是会立即扩展定义，而不是在调用时。类似于% define 和% assign 之间的交叉。“ x”后缀代表“ eXpand”，所以“ xdefine”可以被解释为“ eXpand-and-define”。因此你可以这样做:

```
%assign
转让百分
比          ofs        0
%macro      arg
% 宏        Arg        1
            %xdefine %1      dword   [esp+ofs]
            % xdefine% 1     Dword   [ esp + ofs ]
            %assign
            转让百分         ofs+4
            比          ofs  Ofs + 4
%endmacro
% endmacro
```

• Changed the place where the expansion of %$name macros are expanded. Now they are converted into ..@ctxnum.name form when detokenizing, so there are no quirks as before when using %$name arguments to macros, in macros etc. For example:

更改了展开% $name 宏的位置。现在，在解除记忆时，它们被转换为.@ ctxnum.name 形式，因此在对宏、宏等使用% $name 参数时，不会像以前那样出现异常。例如:

```
%macro  abc      1
% 宏 abc 1
        %define %1 hello
        % 定义% 1 hello
%endm
% endm

abc     %$here
这里
%$here
这里
```

Now last line will be expanded into "hello" as expected. This also allows for lots of goodies, a good example are extended "proc" macros included in this archive.
现在，最后一行将如预期的那样扩展为“ hello”。这也允许许许多多的好东西，一个很好的例子是扩展的“ proc”宏包括在这个档案。

• Added a check for "cstk" in smacro_defined() before calling get_ctx() – this allows for things like:
在调用 get _ ctx ()之前，添加了 smacro _ defined ()中的“ cstk”检查——这允许以下操作:

```
%ifdef %$abc
% ifdef% $abc
%endif
% endif
```

to work without warnings even in no context.
在没有警告的情况下工作，甚至在没有上下文的情况下。

• Added a check for "cstk" in %if*ctx and %elif*ctx directives – this allows to use `%ifctx` without excessive warnings. If there is no active context, `%ifctx` goes through "false" branch.
在% if * ctx 和% elif * ctx 指令中添加了“ cstk”检查-这允许在没有过多警告的情况下使用% ifctx。如果没有活动上下文,% ifctx 将通过“ false”分支。

• Removed "user error: " prefix with `%error` directive: it just clobbers the output and has absolutely no functionality. Besides, this allows to write macros that does not differ from built−in functions in any way.
删除“用户错误:”前缀与% error 指令: 它只是打击输出，绝对没有任何功能。此外，这允许编写与内置函数没有任何区别的宏。

259

- Added expansion of string that is output by `%error` directive. Now you can do things like:
添加了由% error 指令输出的字符串扩展。现在您可以执行以下操作:

```
    %define hello(x) Hello, x!
    % define Hello (x) Hello，x!

    %define %$name andy
    % define% $name andy
    %error "hello(%$name)"
    % 错误“ hello (% $name)”
```

  Same happened with `%include` directive.
  % include 指令的头文件也发生了同样的情况。

- Now all directives that expect an identifier will try to expand and concatenate everything without whitespaces in between before usage. For example, with "unfixed" nasm the commands
现在所有需要标识符的指令都会在使用之前尝试扩展和连接所有没有空格的内容。例如，使用“ unfixed” 命令

```
    %define %$abc hello
    % define% $abc hello
    %define __%$abc goodbye
    % define _ _% $abc goodbye
    __%$abc
    百分之百美元
```

  would produce "incorrect" output: last line will expand to
  将产生“不正确”的输出: 最后一行将扩展到

```
    hello goodbyehello
    你好，再见，你好
```

  Not quite what you expected, eh? :−) The answer is that preprocessor treats the `%define` construct as if it would be
  和你想象的不太一样，是吧？答案是，预处理器对待% define 结构就像它本来应该是的那样

```
    %define __ %$abc goodbye
    % define _% $abc goodbye
```

  (note the white space between __ and %$abc). After my "fix" it will "correctly" expand into
  (注意 _ 和% $abc 之间的空格)。在我的“修复”之后，它将“正确”扩展到

```
    goodbye
    再见
```

  as expected. Note that I use quotes around words "correct", "incorrect" etc because this is rather a feature not a bug; however current behaviour is more logical (and allows more advanced macro usage :−).
  不出所料。请注意，我在“正确”、“不正确”等词前面使用了引号，因为这是一个特性，而不是一个 bug; 然而，当前的行为更符合逻辑(并允许更高级的宏用法: -)。

Same change was applied to: `%push`,`%macro`,`%imacro`,`%define`,`%idefine`,`%xdefine`,`%ixdefine`, `%assign`,`%iassign`, `%undef`
同样的更改应用于:% push,% macro,% imacros,% define,% idedefine,% xdefine,% ixdefine,% assign,% iassign,% undef

- A new directive [WARNING {+|−}warning−id] have been added. It works only if the assembly phase is enabled (i.e. it doesn't work with nasm –e).
添加了一个新的指令[WARNING { + |-} WARNING-id]。它只有在组装阶段启用的情况下才能工作(也就是说，它不能在 nasm-e 中工作)。

- A new warning type: macro−selfref. By default this warning is disabled; when enabled NASM warns when a macro self−references itself; for example the following source:

一种新的警告类型: macroself。默认情况下，这个警告是禁用的; 当启用 NASM 警告时，宏自引用本身; 例如以下来源:

```
[WARNING macro-selfref]
[警告宏自由]

%macro          push    1-*
% 宏            推      1-*
        %rep
        % rep   %0
        百分比  % 0
                push    %1
                推      % 1
                %rotate
                % 旋转  1
        %endrep
        百分比
%endmacro
% endmacro

                push    eax,ebx,ecx
                按 eax，ebx，ecx
```

will produce a warning, but if we remove the first line we won’t see it anymore (which is The Right Thing To Do {tm} IMHO since C preprocessor eats such constructs without warnings at all).
将产生一个警告，但如果我们删除第一行，我们将不会再看到它(这是正确的事情做{ tm } IMHO，因为 c 预处理器吃这样的构造没有任何警告)。

- Added a "error" routine to preprocessor which always will set ERR_PASS1 bit in severity_code. This removes annoying repeated errors on first and second passes from preprocessor.

为预处理器添加了一个“错误”例程，该例程将始终在 severity _ 代码中设置 ERR _ pass1 位。这样可以在预处理器的第一次和第二次传递中消除恼人的重复错误。

- Added the %+ operator in single−line macros for concatenating two identifiers. Usage example:

在单行宏中添加% + 操作符以连接两个标识符。用法示例:

260

```
%define _myfunc _otherfunc
% define _ myfunc _ otherfunc
%define cextern(x) _ %+ x
% 定义 cextern (x) _% + x
cextern (myfunc)
Cextern (myfunc)
```

After first expansion, third line will become "_myfunc". After this expansion is performed again so it becomes "_otherunc".
第一次扩展后，第三行将变成“ _ myfunc”。再次执行这个扩展后，它将变成“ _ otherunc”。

• Now if preprocessor is in a non−emitting state, no warning or error will be emitted. Example:
如果预处理器处于非发射状态，不会发出警告或错误。示例:

```
%if 1
% (如 1)
        mov     eax,ebx
        Mov eax，ebx
%else
其他%
        put anything you want between these two brackets,
        even macro-parameter references %1 or local
        labels %$zz or macro-local labels %%zz - no
        warning will be emitted.
        在这两个括号之间放置任何内容，即使是宏参数引用% 1 或本地标
        签% $zz 或宏本地标签%% zz，也不会发出警告。
%endif
% endif
```

• Context−local variables on expansion as a last resort are looked up in outer contexts. For example, the following piece:
Context-在外部环境中查找作为最后手段的扩展局部变量。例如，下面这段:

```
%push   outer
百分比推出
%define %$a [esp]
% define% $a [ esp ]

        %push   inner
        百分之推进内
        %$a
        % $a
        %pop
        百分之一
%pop
百分之一
```

will expand correctly the fourth line to [esp]; if we’ll define another %$a inside the "inner" context, it will take precedence over outer definition. However, this modification has been applied only to expand_smacro and not to smacro_define: as a consequence expansion looks in outer contexts, but `%ifdef` won’t look in outer contexts.
将正确地将第四行扩展到[esp] ; 如果我们在“内部”上下文中定义另一个% $a，它将优先于外部定义。然而，这个修改只应用于展开 _ 宏，而不应用于宏 _ define: 因此展开查看外部上下文，但% ifdef 不查看外部上下文。

This behaviour is needed because we don’t want nested contexts to act on already defined local macros. Example:
这种行为是必要的，因为我们不希望嵌套上下文对已经定义好的本地宏起作用。示例:

```
        %$arg1    [esp+4]
%define % $arg1   [ esp + 4]
```

```
百分比定
义
          eax,eax
test      前进，前
测试      进
if        nz
如果      新西兰
          mov       eax,%$arg1
          动起来    Eax,% $arg1
endif
译注:
```

In this example the "if" mmacro enters into the "if" context, so %$arg1 is not valid anymore inside "if". Of course it could be worked around by using explicitely %$$arg1 but this is ugly IMHO.
在本例中，“ if”mmacro 进入“ if”上下文，因此% $arg1 在“ if”中不再有效。当然，可以通过显式地使用% $$arg1 来解决这个问题，但是这太难看了。

- Fixed memory leak in `%undef`. The origline wasn't freed before exiting on success.

修正了% unf 中的内存泄漏。原始版本在成功退出之前没有被释放。

- Fixed trap in preprocessor when line expanded to empty set of tokens. This happens, for example, in the following case:

当行扩展为空的令牌集时，预处理器中的固定陷阱。例如，在以下情况下会发生这种情况:

```
#define SOMETHING
定义一些东西
SOMETHING
一些东西
```

C.2.41 Version 0.98
C. 2.41 版本 0.98

All changes since NASM 0.98p3 have been produced by H. Peter Anvin <hpa@zytor.com>.
自 NASM 0.98 p3 以来的所有变化均由 h. Peter Anvin < hpa@zytor. com > 完成。

- The documentation comment delimiter is

文档注释分隔符是

- Allow EQU definitions to refer to external labels; reported by Pedro Gimeno.

允许 EQU 定义参考外部标签; 由 Pedro Gimeno 报道。

- Re−enable support for RDOFF v1; reported by Pedro Gimeno.
重新启用对 RDOFF v1 的支持; Pedro Gimeno 报道。

- Updated License file per OK from Simon and Julian.
来自 Simon 和 Julian 的更新许可文件。

C.2.42 Version 0.98p9
C. 2.42 版本 0.98 p9

- Update documentation (although the instruction set reference will have to wait; I don't want to hold up the 0.98 release for it.)
更新文档(虽然指令集引用需要等待; 我不想耽误 0.98 版本)

- Verified that the NASM implementation of the PEXTRW and PMOVMSKB instructions is correct. The encoding differs from what the Intel manuals document, but the Pentium III behaviour matches NASM, not the Intel manuals.
验证了 PEXTRW 和 PMOVMSKB 指令的 NASM 实现是正确的。编码不同于英特尔手册文档，但奔腾 III 的行为匹配 NASM，而不是英特尔手册。

- Fix handling of implicit sizes in PSHUFW and PINSRW, reported by Stefan Hoffmeister.
修正了 PSHUFW 和 PINSRW 中隐式大小的处理，由 Stefan Hoffmeister 报道。

- Resurrect the –s option, which was removed when changing the diagnostic output to stdout.
重新启动 -s 选项，该选项在将诊断输出更改为 stdout 时被删除。

C.2.43 Version 0.98p8
C. 2.43 版本 0.98 p8

- Fix for "DB" when NASM is running on a bigendian machine.
当 NASM 在 bigendian 机器上运行时修复“ DB”。

- Invoke insns.pl once for each output script, making Makefile.in legal for "make –j".
对每个输出脚本调用 insns.pl 一次，使 Makefile.in 合法化为“ make-j”。

- Improve the Unix configure−based makefiles to make package creation easier.
改进基于 Unix 配置的 makefile，使包的创建更加容易。

- Included an RPM .spec file for building RPM (RedHat Package Manager) packages on Linux or Unix systems.
包括一个 RPM。 spec 文件，用于在 Linux 或 Unix 系统上构建 RPM (RedHat Package Manager) 包。

- Fix Makefile dependency problems.
修正 Makefile 依赖性问题。

- Change src/rdsrc.pl to include sectioning information in info output; required for install−info to work.
更改 src/rdsrc.pl，在 info 输出中包含分段信息; install-info 工作所需。

- Updated the RDOFF distribution to version 2 from Jules; minor massaging to make it compile in my environment.
从 Jules 将 RDOFF 发行版更新为版本 2; 进行少量消息处理，使其在我的环境中编译。

- Split doc files that can be built by anyone with a Perl interpreter off into a separate archive.
Split 文档文件，任何人都可以通过 Perl 解释器将其构建到一个单独的存档中。

- "Dress rehearsal" release!
”彩排”发布！

C.2.44 Version 0.98p7
C. 2.44 Version 0.98 p7

- Fixed opcodes with a third byte−sized immediate argument to not complain if given "byte" on the immediate.
修正了带有第三个字节大小的即时参数的操作码，如果在即时参数上给定“ byte”，则不会抱怨。

- Allow `%undef` to remove single−line macros with arguments. This matches the behaviour of #undef in the C preprocessor.

允许% undef 删除带参数的单行宏。这与 c 预处理器中 # undef 的行为相匹配。

- Allow –d, –u, –i and –p to be specified as –D, –U, –I and –P for compatibility with most C compilers and preprocessors. This allows Makefile options to be shared between cc and nasm, for example.

为了与大多数 c 编译器和预处理器兼容，允许将 -d、-u、-i 和 -p 指定为 -d、-u、-i 和 -p。例如，这允许在 cc 和 nasm 之间共享 Makefile 选项。

- Minor cleanups.

小的清理。

- Went through the list of Katmai instructions and hopefully fixed the (rather few) mistakes in it.

仔细阅读了 Katmai 的说明清单，希望能够修正其中的错误(相当少)。

- (Hopefully) fixed a number of disassembler bugs related to ambiguous instructions (disambiguated by –p) and SSE instructions with REP.

修正了一些反汇编程序的错误，这些错误与模棱两可的指令(被 -p 消除了歧义)和使用 rep 的 SSE 指令有关。

- Fix for bug reported by Mark Junger: "call dword 0x12345678" should work and may add an OSP (affected CALL, JMP, Jcc).

修复由 Mark Junger 报告的 bug: “ CALL dword 0x12345678”应该可以工作，并可能添加一个 OSP (受影响的 CALL，JMP，Jcc)。

- Fix for environments when "stderr" isn’t a compile−time constant.

修正当“ stderr”不是编译时常量时的环境。

C.2.45 Version 0.98p6
C. 2.45 版本 0.98 p6

- Took officially over coordination of the 0.98 release; so drop the p3.x notation. Skipped p4 and p5 to avoid confusion with John Fine’s J4 and J5 releases.

正式接管 0.98 版本的协调工作; 所以放弃 p3.x 符号。跳过 p4 和 p5 以避免与 John Fine 的 j4 和 j5 版本混淆。

- Update the documentation; however, it still doesn't include documentation for the various new instructions. I somehow wonder if it makes sense to have an instruction set reference in the assembler manual when Intel et al have PDF versions of their manuals online.

更新文档; 然而，它仍然不包括各种新指令的文档。我想知道，当 Intel 等公司的手册在线有 PDF 版本时，在汇编手册中引用一个指令集是否有意义。

- Recognize "idt" or "centaur" for the –p option to ndisasm.

识别“ idt”或“ centaur”作为 ndisasm 的 -p 选项。

- Changed error messages back to stderr where they belong, but add an –E option to redirect them elsewhere (the DOS shell cannot redirect stderr.)

已更改的错误消息返回到它们所属的 stderr，但是添加了 -e 选项将它们重定向到其他地方(DOS shell 不能重定向 stderr。)

- –M option to generate Makefile dependencies (based on code from Alex Verstak.)

- m 选项来生成 Makefile 依赖项(基于 Alex Verstak 的代码)

- `%undef` preprocessor directive, and –u option, that undefines a single−line macro.

`% undef` 预处理器指令和 `-u` 选项，取消单行宏的定义。

- OS/2 Makefile (Mkfiles/Makefile.os2) for Borland under OS/2; from Chuck Crayne.

OS/2 下 Borland 的 OS/2 Makefile (Mkfiles/Makefile.os2) ; 来自 Chuck Crayne。

- Various minor bugfixes (reported by): – Dangling `%s` in preproc.c (Martin Junker)

各种各样的小错误修复(报道) :-Dangling% s in preproc.c (Martin Junker)

- THERE ARE KNOWN BUGS IN SSE AND THE OTHER KATMAI INSTRUCTIONS. I am on a trip and didn't bring the Katmai instruction reference, so I can't work on them right now.

在 SSE 和其他 KATMAI 指令中有已知的 bug。我正在旅行，没有带 Katmai 指令参考，所以我现在不能处理它们。

- Updated the License file per agreement with Simon and Jules to include a GPL distribution clause.

根据与 Simon 和 Jules 的协议更新了许可证文件，包括一个 GPL 分配条款。

C.2.46 Version 0.98p3.7
C. 2.46 版本 0.98 p3.7

- (Hopefully) fixed the canned Makefiles to include the outrdf2 and zoutieee modules.

(但愿如此)修复了包含 outdf2 和 zoutieee 模块的罐装 Makefiles。

- Renamed changes.asm to changed.asm.

改名为 changes.asm 改名为 chang.asm。

C.2.47 Version 0.98p3.6
C. 2.47 Version 0.98 p3.6

- Fixed a bunch of instructions that were added in 0.98p3.5 which had memory operands, and the address−size prefix was missing from the instruction pattern.

修正了 0.98 p3.5 中添加的一系列指令，这些指令有内存操作数，并且指令模式中缺少地址大小前缀。

C.2.48 Version 0.98p3.5
C. 2.48 Version 0.98 p3.5

- Merged in changes from John S. Fine's 0.98−J5 release. John's based 0.98−J5 on my 0.98p3.3 release; this merges the changes.

融合了 John s. Fine 的 0.98-j5 版本的更改。John 的 0.98-j5 基于我的 0.98 p3.3 版本; 这融合了变化。

- Expanded the instructions flag field to a long so we can fit more flags; mark SSE (KNI) and AMD or Katmai−specific instructions as such.

将 instructions 标志字段扩展为 long，这样我们就可以容纳更多的标志; 标记 SSE (KNI)和 AMD 或 Katmai 特定的指令。

- Fix the "PRIV" flag on a bunch of instructions, and create new "PROT" flag for protected−mode−only instructions (orthogonal to if the instruction is privileged!) and new "SMM" flag for SMM−only instructions.

在一堆指令上固定“ PRIV”标志，并为仅保护模式的指令创建新的“ PROT”标志(与指令是否具有特权正交!)和新的“ SMM”标志为 SMM-only 指令。

- Added AMD−only SYSCALL and SYSRET instructions.

增加了 AMD-only SYSCALL 和 SYSRET 指令。

- Make SSE actually work, and add new Katmai MMX instructions.

使 SSE 实际工作，并添加新的 Katmai MMX 指令。

- Added a –p (preferred vendor) option to ndisasm so that it can distinguish e.g. Cyrix opcodes also used in SSE. For example:

为 ndisasm 添加了一个 -p (首选供应商)选项，这样它就可以区分例如在 SSE 中也使用的 Cyrix 操作码。例如:

```
ndisasm −p cyrix aliased.bin
Ndisasm-p cyrix 别名.bin
          670F514310       paddsiw mm0,[ebx+0x10]
00000000  670F514310       Paddsiw mm0[ ebx + 0 x10]
                           paddsiw mm0,[ebx+0x20]
          670F514320       Paddsiw mm0，[ ebx + 0
00000005  670F514320       x20]
ndisasm −p intel aliased.bin
Ndisasm-p 英特尔别名.bin
          670F514310       sqrtps xmm0,[ebx+0x10]
00000000  670F514310       Sqrtps xmm0，[ ebx + 0 x10]
          670F514320       sqrtps xmm0,[ebx+0x20]
00000005  670F514320       Sqrtps xmm0，[ ebx + 0 x20]
```

- Added a bunch of Cyrix−specific instructions.

•添加了一系列针对 Cyrix 的说明。

C.2.49 Version 0.98p3.4

C. 2.49 版本 0.98 p3.4

- Made at least an attempt to modify all the additional Makefiles (in the Mkfiles directory). I can’t test it, but this was the best I could do.

至少尝试修改所有附加的 Makefiles (在 Mkfiles 目录中)。我不能测试它，但这是我能做的最好的。

- DOS DJGPP+"Opus Make" Makefile from John S. Fine.

DOS DJGPP + “ Opus Make”Makefile from John s. Fine。

- changes.asm changes from John S. Fine.

来自 John S.的 changes.asm 更改。

C.2.50 Version 0.98p3.3
C. 2.50 版本 0.98 p3.3

- Patch from Conan Brink to allow nesting of `%rep` directives.

来自 Conan Brink 的补丁允许嵌套% rep 指令。

- If we’re going to allow INT01 as an alias for INT1/ICEBP (one of Jules 0.98p3 changes), then we should allow INT03 as an alias for INT3 as well.

如果我们允许 int01 作为 INT1/ICEBP 的别名(Jules 0.98 p3 变化之一)，那么我们也应该允许 int03 作为 int3 的别名。

- Updated changes.asm to include the latest changes.

更新了 changes.asm 以包含最新的更改。

- Tried to clean up the <CR>s that had snuck in from a DOS/Windows environment into my Unix environment, and try to make sure than DOS/Windows users get them back.

试图清理从 DOS/Windows 环境中潜入到我的 Unix 环境中的 < cr >，并试图确保 DOS/Windows 用户能够找回它们。

- We would silently generate broken tools if insns.dat wasn’t sorted properly. Change insns.pl so that the order doesn’t matter.

如果 insns.dat 没有被正确地排序，我们会默默地生成坏掉的工具。修改 insns.pl，这样顺序就不重要了。

- Fix bug in insns.pl (introduced by me) which would cause conditional instructions to have an extra "cc" in disassembly, e.g. "jnz" disassembled as "jccnz".

Pl 中的 Fix bug (由我介绍)，它会导致条件指令在反汇编中有一个额外的“ cc”，例如将“ jnz”反汇编为“ jccnz”。

C.2.51 Version 0.98p3.2
C. 2.51 Version 0.98 p3.2

- Merged in John S. Fine’s changes from his 0.98−J4 prerelease; see http://www.csoft.net/cz/johnfine/

合并到 John s. Fine 的 0.98-j4 预发布版中的更改中; 参见 http://www.csoft. net/cz/johnfine/

- Changed previous "spotless" Makefile target (appropriate for distribution) to "distclean", and added "cleaner" target which is same as "clean" except deletes files generated by Perl scripts; "spotless" is union.

将以前的“一尘不染”Makefile 目标(适合分发)更改为“ distclean”，并添加了与“ clean”相同的“ cleaner”目标，除了删除由 Perl 脚本生成的文件外; “一尘不染”是 union。

- Removed BASIC programs from distribution. Get a Perl interpreter instead (see below.)

从发行版中删除 BASIC 程序，改用 Perl 解释器(参见下文)

- Calling this "pre−release 3.2" rather than "p3−hpa2" because of John’s contributions.

因为 John 的贡献，我们称之为“ pre-release 3.2”而不是“ p3-hpa2”。

- Actually link in the IEEE output format (zoutieee.c); fix a bunch of compiler warnings in that file. Note I don’t know what IEEE output is supposed to look like, so these changes were made "blind".

实际上以 IEEE 输出格式(zoutieee.c)链接; 修复该文件中的一系列编译器警告。注意: 我不知道 IEEE 的输出应该是什么样子，所以这些修改是“盲”的。

C.2.52 Version 0.98p3−hpa
C. 2.52 Version 0.98 p3-hpa

- Merged nasm098p3.zip with nasm−0.97.tar.gz to create a fully buildable version for Unix systems (Makefile.in updates, etc.)

将 nasm098p3.zip 与 nasm-0.97.tar.gz 合并，为 Unix 系统创建一个完全可构建的版本(Makefile.in updates，etc)

• Changed insns.pl to create the instruction tables in nasm.h and names.c, so that a new instruction can be added by adding it *only* to insns.dat.

修改 insns.pl 以创建 nasm.h 和 names.c 中的指令表，这样就可以通过将 * only * 添加到 insns.dat 中来添加新的指令。

• Added the following new instructions: SYSENTER, SYSEXIT, FXSAVE, FXRSTOR, UD1, UD2 (the latter two are two opcodes that Intel guarantee will never be used; one of them is documented as UD2 in Intel documentation, the other one just as "Undefined Opcode" — calling it UD1 seemed to make sense.)

添加了以下新指令: SYSENTER、 SYSEXIT、 FXSAVE、 FXRSTOR、 UD1、 UD2(后两个操作码是 Intel 保证永远不会使用的操作码; 其中一个在 Intel 文档中被记录为 UD2，另一个只是“ Undefined Opcode”——称之为 ud1 似乎有道理。)

• MAX_SYMBOL was defined to be 9, but LOADALL286 and LOADALL386 are 10 characters long. Now MAX_SYMBOL is derived from insns.dat.

MAX _ symbol 被定义为 9，但 loadall286 和 loadall386 只有 10 个字符长。现在 MAX _ symbol 来源于 insns.dat。

• A note on the BASIC programs included: forget them. insns.bas is already out of date. Get yourself a Perl interpreter for your platform of choice at http://www.cpan.org/ports/index.html.

关于 BASIC 程序的注释包括: 忘记它们。Insns.bas 已经过时了。在 http://www.cpan.org/ports/index. html 为你的平台选择一个 Perl 解释器。

C.2.53 Version 0.98 pre−release 3

C. 2.53 Version 0.98 pre-release 3

• added response file support, improved command line handling, new layout help screen

增加了响应文件支持，改进了命令行处理，新的布局帮助屏幕

• fixed limit checking bug, ’OUT byte nn, reg’ bug, and a couple of rdoff related bugs, updated Wishlist; 0.98 Prerelease 3.

固定的限制检查错误,’OUT 字节 nn，reg’错误，和一对夫妇的 rdoff 相关的错误，更新 Wishlist; 0.98 预发布 3。

264

C.2.54 Version 0.98 pre−release 2
C. 2.54 Version 0.98 pre-release 2 c. 2.54 Version 0.98 pre-release 2

- fixed bug in outcoff.c to do with truncating section names longer than 8 characters, referencing beyond end of string; 0.98 pre−release 2

修正了 outcoff.c 中的 bug，用于截断超过 8 个字符的节名，引用超过字符串的末尾; 0.98 pre-release 2

C.2.55 Version 0.98 pre−release 1
C. 2.55 Version 0.98 pre-release 1

- Fixed a bug whereby STRUC didn't work at all in RDF.

修正了 STRUC 在 RDF 中根本无法工作的错误。

- Fixed a problem with group specification in PUBDEFs in OBJ.

修正了 OBJ 中 PUBDEFs 中的组规范问题。

- Improved ease of adding new output formats. Contribution due to Fox Cutter.

改进了添加新输出格式的容易性。

- Fixed a bug in relocations in the 'bin' format: was showing up when a relocatable reference crossed an 8192−byte boundary in any output section.

修正了一个" bin"格式的重定位错误: 当一个可重定位的引用在任何输出部分跨越 8192 字节边界时出现错误。

- Fixed a bug in local labels: local−label lookups were inconsistent between passes one and two if an EQU occurred between the definition of a global label and the subsequent use of a local label local to that global.

修正了本地标签中的一个错误: 如果在全局标签的定义和随后使用该全局标签的本地标签之间发生了 EQU，那么本地标签查找在第一次和第二次之间是不一致的。

- Fixed a seg−fault in the preprocessor (again) which happened when you use a blank line as the first line of a multi−line macro definition and then defined a label on the same line as a call to that macro.

修正了预处理器中的一个分段错误(再次)，这个错误发生在您使用一个空行作为多行宏定义的第一行，然后在同一行定义一个标签作为对该宏的调用。

- Fixed a stale−pointer bug in the handling of the NASM environment variable. Thanks to Thomas McWilliams.

修正了 NASM 环境变量处理过时的指针错误。

- ELF had a hard limit on the number of sections which caused segfaults when transgressed. Fixed.

ELF 有一个硬限制，当违反时会导致段错误。 Fixed。

- Added ability for ndisasm to read from stdin by using '−' as the filename.

增加了 ndisasm 通过使用'-'作为文件名从 stdin 读取的能力。

- ndisasm wasn't outputting the TO keyword. Fixed.

Ndisasm 没有输出 TO 关键字。

- Fixed error cascade on bogus expression in `%if` – an error in evaluation was causing the entire `%if` to be discarded, thus creating trouble later when the `%else` or `%endif` was encountered.

修正了如果计算中的错误导致丢弃整个%，则在% 中假表达式上的错误级联，从而在以后遇到% else 或% endif 时造成麻烦。

- Forward reference tracking was instruction−granular not operand− granular, which was causing 286−specific code to be generated needlessly on code of the form 'shr word [forwardref],1'. Thanks to Jim Hague for sending a patch.

正向引用跟踪是指令粒度而不是操作数粒度的，这导致不必要地在' shr word [foredref]，1'形式的代码上生成 286 个特定代码。感谢 Jim Hague 发送了一个补丁。

- All messages now appear on stdout, as sending them to stderr serves no useful purpose other than to make redirection difficult.

所有消息现在都出现在 stdout 上，因为将它们发送到 stderr 除了使重定向变得困难之外，没有其他有用的目的。

- Fixed the problem with EQUs pointing to an external symbol – this now generates an error message.

修正了 EQUs 指向一个外部符号的问题——这会生成一个错误消息。

- Allowed multiple size prefixes to an operand, of which only the first is taken into account.

允许操作数的多个大小前缀，其中只考虑第一个前缀。

- Incorporated John Fine's changes, including fixes of a large number of preprocessor bugs, some small problems in OBJ, and a reworking of label handling to define labels before their line is assembled, rather than after.

合并 John Fine 的改变，包括修复大量的预处理器错误，OBJ 中的一些小问题，以及重新工作的标签处理，以定义标签在他们的行组装之前，而不是之后。

- Reformatted a lot of the source code to be more readable. Included 'coding.txt' as a guideline for how to format code for contributors.

重新格式化了许多源代码，使其更易于阅读。其中包括“ coding.txt”作为指导如何为贡献者格式化代码。

- Stopped nested `%reps` causing a panic – they now cause a slightly more friendly error message instead.

停止嵌套% reps 引起恐慌——它们现在会导致一个稍微友好一点的错误消息。

- Fixed floating point constant problems (patch by Pedro Gimeno)

固定浮点常数问题(Pedro Gimeno 修补程序)

- Fixed the return value of insn_size() not being checked for –1, indicating an error.

修正了 insn _ size ()的返回值没有被检查为 -1，表示有错误。

- Incorporated 3Dnow! instructions.

公司 3dnow! 说明书。

- Fixed the 'mov eax, eax + ebx' bug.

修正了“ mov eax，eax + ebx”错误。

- Fixed the GLOBAL EQU bug in ELF. Released developers release 3.

修正了 ELF 中的 GLOBAL EQU 错误。发布了开发者版本 3。

- Incorporated John Fine’s command line parsing changes
合并 John Fine 的命令行解析更改

- Incorporated David Lindauer’s OMF debug support
合并 David Lindauer 的 OMF 调试支持

- Made changes for LCC 4.0 support (`__NASM_CDecl__`, removed register size specification warning when sizes agree).
对 lcc4.0 支持进行了更改(_ _ NASM _ cdecl _ _，在大小一致时删除了寄存器大小规范警告)。

C.3 NASM 0.9 Series
C. 3 NASM 0.9 系列

Revisions before 0.98.
0.98 之前的修订。

C.3.1 Version 0.97 released December 1997
C. 3.1 版本 0.971997 年 12 月发布

- This was entirely a bug−fix release to 0.96, which seems to have got cursed. Silly me.
这完全是一个 0.96 版本的 bug 修复版本，它似乎被诅咒了。我真傻。

- Fixed stupid mistake in OBJ which caused ‘MOV EAX,<constant>’ to fail. Caused by an error in the ‘MOV EAX,<segment>’ support.
修正了 OBJ 中导致“ MOV EAX，< constant >”失败的愚蠢错误。这是由于“ MOV EAX，< segment >”支持中的错误造成的。

- ndisasm hung at EOF when compiled with lcc on Linux because lcc on Linux somehow breaks feof(). ndisasm now does not rely on feof().
在 Linux 上用 lcc 编译时，ndisasm 挂在 EOF 上，因为 Linux 上的 lcc 在某种程度上破坏了 feof ()。Ndisasm 现在不依赖于 feof ()。

- A heading in the documentation was missing due to a markup error in the indexing. Fixed.
文档中的一个标题由于索引中的标记错误而丢失。修正。

- Fixed failure to update all pointers on realloc() within extended− operand code in parser.c. Was causing wrong behaviour and seg faults on lines such as ‘dd 0.0,0.0,0.0,0.0,...’
修正了在 parser.c 的扩展操作数代码中更新 realloc ()上的所有指针的错误。在线路上造成错误行为和断层错误，比如‘ dd 0.0,0.0,0.0，...’

- Fixed a subtle preprocessor bug whereby invoking one multi−line macro on the first line of the expansion of another, when the second had been invoked with a label defined before it, didn’t expand the inner macro.
修正了一个微妙的预处理器错误，即在另一个扩展的第一行调用一个多行宏，而在第二行调用之前定义了一个标签，不会扩展内部宏。

- Added internal.doc back in to the distribution archives – it was missing in 0.96 *blush*
将 internal.doc 添加回发布文档中——它在 0.96 * blush * 中丢失

- Fixed bug causing 0.96 to be unable to assemble its own test files, specifically objtest.asm. *blush again*
修正了导致 0.96 无法自己组装测试文件的错误，特别是 objtest.asm。 * 腮红再次 *

- Fixed seg−faults and bogus error messages caused by mismatching `%rep` and `%endrep` within multi−line macro definitions.
修正了多行宏定义中由于% rep 和% endrep 不匹配而导致的分段错误和假错误消息。

- Fixed a problem with buffer overrun in OBJ, which was causing corruption at ends of long PUBDEF records.
修正了 OBJ 中的缓冲区溢出问题，该问题导致了长 PUBDEF 记录末端的损坏。

- Separated DOS archives into main−program and documentation to reduce download size.
分离 DOS 档案到主程序和文档，以减少下载大小。

C.3.2 Version 0.96 released November 1997
C. 3.2 版本 0.96 发布于 1997 年 11 月

- Fixed a bug whereby, if ‘nasm sourcefile’ would cause a filename collision warning and put output into ‘nasm.out’, then ‘nasm sourcefile –o outputfile’ still gave the warning even though the ‘−o’ was honoured. Fixed name pollution under Digital UNIX: one of its header files defined R_SP, which broke the enum in nasm.h.

修正了一个错误，如果‘ nasm sourcefile’会导致文件名冲突警告，并将输出输入到‘ nasm.out’，那么‘ nasm sourcefile-o outputfile’仍然会发出警告，即使‘ o’已被遵守。修正了 Digital UNIX 下的名称污染: 其中一个头文件定义了 r _ sp，破坏了 nasm.h 中的枚举。

- Fixed minor instruction table problems: FUCOM and FUCOMP didn’t have two−operand forms; NDISASM didn’t recognise the longer register forms of PUSH and POP (eg FF F3 for PUSH BX); TEST mem,imm32 was flagged as undocumented; the 32−bit forms of CMOV had 16−bit operand size prefixes; ‘AAD imm’ and ‘AAM imm’ are no longer flagged as undocumented because the Intel Architecture reference documents them.

修正了较小的指令表问题: FUCOM 和 FUCOMP 没有两个操作数形式; NDISASM 没有识别 PUSH 和 POP 较长的寄存器形式(例如 PUSH BX 的 FF F3) ; TEST mem，imm32 被标记为未记录; 32 位 CMOV 有 16 位操作数大小前缀; ‘ AAD imm’和‘ AAM imm’不再被标记为未记录，因为 Intel Architecture 参考文档记录了它们。

- Fixed a problem with the local−label mechanism, whereby strange types of symbol (EQUs, auto−defined OBJ segment base symbols) interfered with the ‘previous global label’ value and screwed up local labels.

修正了本地标签机制的一个问题，即奇怪的符号类型(EQUs，自动定义的 OBJ 段基本符号)干扰了“以前的全局标签”的值，并且弄乱了本地标签。

- Fixed a bug whereby the stub preprocessor didn’t communicate with the listing file generator, so that the –a and –l options in conjunction would produce a useless listing file.

修正了存根预处理器无法与列表文件生成器通信的错误，因此 -a 和 -l 选项结合起来会生成一个无用的列表文件。

- Merged ‘os2’ object file format back into ‘obj’, after discovering that ‘obj’ _also_ shouldn’t have a link pass separator in a module containing a non−trivial MODEND. Flat segments are now declared using the FLAT attribute. ‘os2’ is no longer a valid object format name: use ‘obj’.

在发现‘ obj’_ 也不应该在包含非平凡 MODEND 的模块中有链路传递分隔符之后，将‘ os2’对象文件格式合并回‘ obj’。FLAT 段现在使用 FLAT 属性声明。Os2 不再是一个有效的对象格式名称: 使用 obj。

- Removed the fixed−size temporary storage in the evaluator. Very very long expressions (like ‘mov ax,1+1+1+1+...’ for two hundred 1s or so) should now no longer crash NASM.

删除计算器中固定大小的临时存储。非常长的表达式(比如 201 秒左右的“ mov ax，1 + 1 + 1 + 1 + ...”)现在应该不会崩溃 NASM 了。

- Fixed a bug involving segfaults on disassembly of MMX instructions, by changing the meaning of one of the operand−type flags in nasm.h. This may cause other apparently unrelated MMX problems; it needs to be tested thoroughly.

通过改变 nasm.h 中一个操作数类型标志的含义，修正了 MMX 指令反汇编时出现的 segfault 错误。这可能会导致其他明显不相关的 MMX 问题; 它需要彻底测试。

- Fixed some buffer overrun problems with large OBJ output files. Thanks to DJ Delorie for the bug report and fix.

修正了一些大 OBJ 输出文件的缓冲区溢出问题。感谢 DJ Delorie 的错误报告和修复。

- Made preprocess−only mode actually listen to the `%line` markers as it prints them, so that it can report errors more sanely.

Made preprocess-only 模式实际上在打印时会监听% line marker，这样它可以更理智地报告错误。

- Re−designed the evaluator to keep more sensible track of expressions involving forward references: can now cope with previously−nightmare situations such as:

重新设计了计算器，以保持对涉及前向引用的表达式更合理的跟踪: 现在可以处理以前的噩梦情况，例如:

```
mov ax,foo |
bar foo equ 1
Mov ax，foo |
bar foo equ 1
bar equ 2
第二小节
```

- Added the ALIGN and ALIGNB standard macros.

添加 ALIGN 和 ALIGN 标准宏。

- Added PIC support in ELF: use of WRT to obtain the four extra relocation types needed.

在 ELF 中添加了 PIC 支持: 使用 WRT 获得所需的四种额外的重定位类型。

- Added the ability for output file formats to define their own extensions to the GLOBAL, COMMON and EXTERN directives.

增加了输出文件格式的能力，以定义自己的扩展到 GLOBAL，COMMON 和 EXTERN 指令。

- Implemented common−variable alignment, and global−symbol type and size declarations, in ELF.

在 ELF 中实现了公共变量对齐以及全局符号类型和大小声明。

- Implemented NEAR and FAR keywords for common variables, plus far−common element size specification, in OBJ.

在 OBJ 中实现了公共变量的 NEAR 和 FAR 关键字，以及远公共元素大小规范。

- Added a feature whereby EXTERNs and COMMONs in OBJ can be given a default WRT specification (either a segment or a group).

增加了一个特性，通过这个特性，OBJ 中的 EXTERNs 和 COMMONs 可以得到一个默认的 WRT 规范(片段或组)。

- Transformed the Unix NASM archive into an auto−configuring package.

将 Unix NASM 归档文件转换为自动配置包。

- Added a sanity−check for people applying SEG to things which are already segment bases: this previously went unnoticed by the SEG processing and caused OBJ−driver panics later.

增加了一个健全性检查，为人们应用 SEG 的东西，已经是段基础: 这以前没有被 SEG 处理忽视，并导致 OBJ-驱动程序后来恐慌。

- Added the ability, in OBJ format, to deal with ‘MOV EAX,<segment>’ type references: OBJ doesn’t directly support dword−size segment base fixups, but as long as the low two bytes of the constant term are zero, a word−size fixup can be generated instead and it will work.

增加了 OBJ 格式的处理“ MOV EAX，< segment >”类型引用的能力: OBJ 不直接支持 dword 大小的段基本修复，但是只要常量项的最小两个字节为零，就可以生成一个字大小的修复，它将工作。

- Added the ability to specify sections’ alignment requirements in Win32 object files and pure binary files.

增加了在 win32 对象文件和纯二进制文件中指定节对齐要求的能力。

- Added preprocess−time expression evaluation: the `%assign` (and `%iassign`) directive and the bare `%if` (and `%elif`) conditional. Added relational operators to the evaluator, for use only in `%if` constructs: the standard relationals = < > <= >= <> (and C−like synonyms == and !=) plus low−precedence logical operators &&, ^^ and ||.

增加了预处理时间表达式计算:% 赋值(和% iassign)指令和裸% if (和% elif)条件。向计算器添加了关系运算符，仅用于构造的% 中: 标准关系 = < > < = > = < > (和类 c 的同义词 = = 和！ =)加上低优先级逻辑运算符 & &，^ ^ 和 | | 。

- Added a preprocessor repeat construct: `%rep` / `%exitrep` / `%endrep`.

添加了一个预处理器重复构造:% rep/% exitrep/% endrep。

- Added the __FILE__ and __LINE__ standard macros.

添加 _ _ FILE _ _ 和 _ _ LINE _ _ 标准宏。

- Added a sanity check for number constants being greater than 0xFFFFFFFF. The warning can be disabled.

添加了数字常量大于 0xffffff 的健全性检查。警告可以被禁用。

- Added the %0 token whereby a variadic multi−line macro can tell how many parameters it's been given in a specific invocation.

添加了% 0 令牌，通过这个令牌，可变的多行宏可以知道在特定调用中给定了多少个参数。

- Added `%rotate`, allowing multi−line macro parameters to be cycled.

Add% rotate，允许多行宏参数循环。

- Added the '*' option for the maximum parameter count on multi−line macros, allowing them to take arbitrarily many parameters.

为多行宏的最大参数计数添加了“ *”选项，允许它们任意选取多个参数。

- Added the ability for the user−level forms of EXTERN, GLOBAL and COMMON to take more than one argument.

增加了 EXTERN、 GLOBAL 和 COMMON 的用户级表单可以接受多个参数的能力。

- Added the IMPORT and EXPORT directives in OBJ format, to deal with Windows DLLs.

添加了 OBJ 格式的 IMPORT 和 EXPORT 指令，以处理 windowsdll。

- Added some more preprocessor `%if` constructs: `%ifidn` / `%ifidni` (exact textual identity), and `%ifid` / `%ifnum` / `%ifstr` (token type testing).

添加了更多的预处理器% if 构造:% ifidn/% ifidni (确切的文本标识)和% ifid/% ifnum/% ifstr (令牌类型测试)。

- Added the ability to distinguish SHL AX,1 (the 8086 version) from SHL AX,BYTE 1 (the 286−and−upwards version whose constant happens to be 1).

增加了区分 SHL AX，1(8086 版本)和 SHL AX，BYTE 1(常量为 1 的 286 及以上版本)的能力。

- Added NetBSD/FreeBSD/OpenBSD's variant of a.out format, complete with PIC shared library features.

增加了 NetBSD/FreeBSD/OpenBSD 的 a.out 格式变体，完成了 PIC 共享库特性。

- Changed NASM's idiosyncratic handling of FCLEX, FDISI, FENI, FINIT, FSAVE, FSTCW, FSTENV, and FSTSW to bring it into line with the otherwise accepted standard. The previous behaviour, though it was a deliberate feature, was a deliberate feature based on a misunderstanding. Apologies for the inconvenience.

改变了 NASM 对 FCLEX、 FDISI、 FENI、 FINIT、 FSAVE、 FSTCW、 FSTENV 和 FSTSW 的特殊处理，使其符合其他公认的标准。之前的行为，虽然是一个故意的特性，是一个基于误解的故意特性。不便之处，敬请原谅。

- Improved the flexibility of ABSOLUTE: you can now give it an expression rather than being restricted to a constant, and it can take relocatable arguments as well.

改进了 ABSOLUTE 的灵活性: 你现在可以给它一个表达式，而不是限制在一个常量内，它还可以接受可重定位的参数。

- Added the ability for a variable to be declared as EXTERN multiple times, and the subsequent definitions are just ignored.

添加了将变量多次声明为 EXTERN 的能力，随后的定义将被忽略。

- We now allow instruction prefixes (CS, DS, LOCK, REPZ etc) to be alone on a line (without a following instruction).

我们现在允许指令前缀(CS，DS，LOCK，REPZ 等)在一行中单独存在(没有下面的指令)。

- Improved sanity checks on whether the arguments to EXTERN, GLOBAL and COMMON are valid identifiers.

改进了 EXTERN、 GLOBAL 和 COMMON 参数是否为有效标识符的健全性检查。

- Added misc/exebin.mac to allow direct generation of .EXE files by hacking up an EXE header using DB and DW; also added test/binexe.asm to demonstrate the use of this. Thanks to Yann Guidon for contributing the EXE header code.

添加 misc/exebin.mac 允许直接生成。使用 DB 和 DW 分解一个 EXE 头文件; 还添加了 test/binexe.asm 来演示如何使用它。感谢 Yann Guidon 贡献的 EXE 头代码。

- ndisasm forgot to check whether the input file had been successfully opened. Now it does. Doh!

Ndisasm 忘记检查输入文件是否已经成功打开。现在它做到了!

- Added the Cyrix extensions to the MMX instruction set.

在 MMX 指令集中添加了 Cyrix 扩展。

- Added a hinting mechanism to allow [EAX+EBX] and [EBX+EAX] to be assembled differently. This is important since [ESI+EBP] and [EBP+ESI] have different default base segment registers.

增加了一个暗示机制，允许[eax + ebx]和[ebx + eax]以不同的方式组装。这是重要的，因为[esi + ebp]和[ebp + esi]具有不同的默认基片段寄存器。

- Added support for the PharLap OMF extension for 4096−byte segment alignment.

增加了对 4096 字节段对齐的 PharLap OMF 扩展的支持。

C.3.3 Version 0.95 released July 1997
C. 3.3 版本 0.95 发布于 1997 年 7 月

- Fixed yet another ELF bug. This one manifested if the user relied on the default segment, and attempted to define global symbols without first explicitly declaring the target segment.

修正了另一个 ELF bug。如果用户依赖于默认段，并且试图定义全局符号而不首先明确声明目标段，那么这个问题就会显现出来。

- Added makefiles (for NASM and the RDF tools) to build Win32 console apps under Symantec C++. Donated by Mark Junker.

添加 makefiles (用于 NASM 和 RDF 工具)来构建 Symantec c + + 下的 win32 控制台应用程序。

- Added ‘macros.bas’ and ‘insns.bas’, QBasic versions of the Perl scripts that convert ‘standard.mac’ to ‘macros.c’ and convert ‘insns.dat’ to ‘insnsa.c’ and ‘insnsd.c’. Also thanks to Mark Junker.

添加了‘ macros.bas’和‘ insns.bas’，这是 Perl 脚本的 QBasic 版本，可以将‘ standard.mac’转换为‘ macros.c’，并将‘ insns.dat’转换为‘ insnsa.c’和‘ insnsd.c’。还要感谢 Mark Junker。

- Changed the diassembled forms of the conditional instructions so that JB is now emitted as JC, and other similar changes. Suggested list by Ulrich Doewich.

改变了条件指令的分解形式，使 JB 现在以 JC 的形式发出，以及其他类似的改变。建议列表由 Ulrich Doewich 提供。

- Added ‘@’ to the list of valid characters to begin an identifier with.

将“@”添加到开始标识符的有效字符列表中。

- Documentary changes, notably the addition of the ‘Common Problems’ section in nasm.doc.

更改，特别是在 nasm.doc 中添加了“常见问题”部分。

- Fixed a bug relating to 32−bit PC−relative fixups in OBJ.

修正了一个与 32 位 PC 相关的 OBJ 修正错误。

- Fixed a bug in perm_copy() in labels.c which was causing exceptions in cleanup_labels() on some systems.

修正了 labels.c 中 perm _ copy ()中的一个 bug，这个 bug 在某些系统中导致了 cleanup _ label () 的异常。

- Positivity sanity check in TIMES argument changed from a warning to an error following a further complaint.

TIMES 参数中的 Positivity 健全性检查从一个警告更改为进一步抱怨后的一个错误。

- Changed the acceptable limits on byte and word operands to allow things like ‘~10111001b’ to work.

修改了字节和单词操作数的可接受限制，允许类似“ ~ 10111001b”的操作。

- Fixed a major problem in the preprocessor which caused seg−faults if macro definitions contained blank lines or comment−only lines.

修正了预处理器中的一个主要问题，如果宏定义包含空行或只有注释的行，则会导致切分错误。

- Fixed inadequate error checking on the commas separating the arguments to ‘db’, ‘dw’ etc.

修正了将参数分隔为“ db”、“ dw”等的逗号错误检查不足的问题。

- Fixed a crippling bug in the handling of macros with operand counts defined with a ‘+’ modifier.

修正了一个在处理带有用“ +”修饰符定义的操作数的宏时的严重缺陷。

- Fixed a bug whereby object file formats which stored the input file name in the output file (such as OBJ and COFF) weren’t doing so correctly when the output file name was specified on the command line.

修正了在命令行中指定输出文件名时，将输入文件名存储在输出文件(如 OBJ 和 COFF)中的对象文件格式不正确的错误。

- Removed [INC] and [INCLUDE] support for good, since they were obsolete anyway.

移除[INC]和[INCLUDE]对好的支持，因为它们已经过时了。

- Fixed a bug in OBJ which caused all fixups to be output in 16−bit (old−format) FIXUPP records, rather than putting the 32−bit ones in FIXUPP32 (new−format) records.

修正了 OBJ 中的一个 bug，这个 bug 导致所有修补程序都以 16 位(旧格式) FIXUPP 记录输出，而不是将 32 位的修补程序放在 fixup32(新格式)记录中。

- Added, tentatively, OS/2 object file support (as a minor variant on OBJ).

暂时添加了 OS/2 对象文件支持(作为 OBJ 的一个小变体)。

- Updates to Fox Cutter’s Borland C makefile, Makefile.bc2.

更新到 Fox Cutter 的 Borland c makefile，Makefile.bc2。

- Removed a spurious second fclose() on the output file.

删除了输出文件上的第二个伪 fclose ()。

- Added the ‘−s’ command line option to redirect all messages which would go to stderr (errors, help text) to stdout instead.

添加了“-s”命令行选项，将所有发送到 stderr (错误，帮助文本)的消息重定向到 stdout。

- Added the ‘−w’ command line option to selectively suppress some classes of assembly warning messages.

添加了“-w”命令行选项，以选择性地禁止某些类别的程序集警告消息。

- Added the ‘−p’ pre−include and ‘−d’ pre−define command−line options.

添加了“-p”pre-include 和“-d”pre-define 命令行选项。

- Added an include file search path: the ‘−i’ command line option.

添加了一个包含文件搜索路径: “-i”命令行选项。

- Fixed a silly little preprocessor bug whereby starting a line with a ‘%!’ environment−variable reference caused an ‘unknown directive’ error.

修正了一个愚蠢的预处理器错误，即以“%”开始一行！环境变量引用导致“未知指令”错误。

- Added the long−awaited listing file support: the ‘−l’ command line option.

添加了期待已久的列表文件支持: “-l”命令行选项。

- Fixed a problem with OBJ format whereby, in the absence of any explicit segment definition, non−global symbols declared in the implicit default segment generated spurious EXTDEF records in the output.

修正了 OBJ 格式的一个问题，即在没有任何显式段定义的情况下，在隐式默认段中声明的非全局符号会在输出中生成虚假的 EXTDEF 记录。

- Added the NASM environment variable.

添加了 NASM 环境变量。

- From this version forward, Win32 console−mode binaries will be included in the DOS distribution in addition to the 16−bit binaries. Added Makefile.vc for this purpose.

从这个版本开始，win32 控制台模式的二进制文件除了 16 位的二进制文件外，还将包含在 DOS 发行版中。为此增加了 Makefile.vc。

- Added ‘return 0;’ to test/objlink.c to prevent compiler warnings.

在 test/objlink.c 中添加了“ return 0;”，以防止编译器发出警告。

- Added the __NASM_MAJOR__ and __NASM_MINOR__ standard defines.

添加了 _ NASM _ major _ _ 和 _ NASM _ minor _ _ _ 标准定义。

269

- Added an alternative memory−reference syntax in which prefixing an operand with ‘&’ is equivalent to enclosing it in square brackets, at the request of Fox Cutter.

增加了一种替代的内存引用语法，其中在操作数前面加上“ &”相当于应 Fox Cutter 的要求将其放在方括号中。

- Errors in pass two now cause the program to return a non−zero error code, which they didn’t before.

第二个错误现在导致程序返回一个非零错误代码，这是他们以前没有做到的。

- Fixed the single−line macro cycle detection, which didn’t work at all on macros with no parameters (caused an infinite loop). Also changed the behaviour of single−line macro cycle detection to work like cpp, so that macros like ‘extrn’ as given in the documentation can be implemented.

修正了单行宏循环检测，这在没有参数的宏上根本不起作用(导致了无限循环)。还改变了单行宏循环检测的行为，使其像 cpp 一样工作，从而可以实现文档中给出的‘ extrn’这样的宏。

- Fixed the implementation of WRT, which was too restrictive in that you couldn’t do ‘mov ax,[di+abc wrt dgroup]’ because (di+abc) wasn’t a relocatable reference.

修正了 WRT 的实现，因为(di + abc)不是一个可重定位的引用，所以不能执行‘ mov ax，[di + abc WRT dgroup]’。

C.3.4 Version 0.94 released April 1997
C. 3.4 版本 0.94 于 1997 年 4 月发布

- Major item: added the macro processor.

主要项目: 添加了宏处理器。

- Added undocumented instructions SMI, IBTS, XBTS and LOADALL286. Also reorganised CMPXCHG instruction into early−486 and Pentium forms. Thanks to Thobias Jones for the information.

添加了无文档说明 SMI，IBTS，XBTS 和 LOADALL286。还将 CMPXCHG 指令重组为早期的 -486 和 Pentium 形式。感谢 Thobias Jones 提供的信息。

- Fixed two more stupid bugs in ELF, which were causing ‘ld’ to continue to seg−fault in a lot of non−trivial cases.

修正了 ELF 中另外两个愚蠢的错误，这些错误会导致‘ ld’在很多非平凡的情况下继续分割错误。

- Fixed a seg−fault in the label manager.

修正了标签管理器中的一个 seg 错误。

- Stopped FBLD and FBSTP from _requiring_ the TWORD keyword, which is the only option for BCD loads/stores in any case.

停止 FBLD 和 FBSTP 从 _ 需要 _ TWORD 关键字，这是唯一的选择为 BCD 加载/存储在任何情况下。

- Ensured FLDCW, FSTCW and FSTSW can cope with the WORD keyword, if anyone bothers to provide it. Previously they complained unless no keyword at all was present.

确保 FLDCW，FSTCW 和 FSTSW 可以处理 WORD 关键字，如果有人麻烦提供它。之前他们抱怨说，除非根本没有关键字。

- Some forms of FDIV/FDIVR and FSUB/FSUBR were still inverted: a vestige of a bug that I thought had been fixed in 0.92. This was fixed, hopefully for good this time...

FDIV/FDIVR 和 FSUB/FSUBR 的一些形式仍然是颠倒的: 一个我认为已经在 0.92 中修复的 bug 的遗迹。这次修好了，希望是永久性的..。

- Another minor phase error (insofar as a phase error can _ever_ be minor) fixed, this one occurring in code of the form

另一个微小的相位错误(只要一个相位错误可以永远是微小的)被修复了，这个发生在表单代码中的错误

```
rol ax,forward_reference
forward_reference equ 1
前向参考前向参考等式 1
```

- The number supplied to TIMES is now sanity−checked for positivity, and also may be greater than 64K (which previously didn’t work on 16−bit systems).

提供给 TIMES 的数字现在已经被检查是否正确，并且可能大于 64k (以前在 16 位系统上不能工作)。

- Added Watcom C makefiles, and misc/pmw.bat, donated by Dominik Behr.

添加了 Watcom c makefiles 和 misc/pmw.bat，由 Dominik Behr 捐赠。

- Added the INCBIN pseudo−opcode.

添加了 INCBIN 伪操作码。

- Due to the advent of the preprocessor, the [INCLUDE] and [INC] directives have become obsolete. They are still supported in this version, with a warning, but won’t be in the next.

由于预处理器的出现，[INCLUDE]和[INC]指令已经过时了。在这个版本中，它们仍然受到支持，有一个警告，但不会在下一个版本中。

- Fixed a bug in OBJ format, which caused incorrect object records to be output when absolute labels were made global.

修正了 OBJ 格式的一个 bug，当绝对标签被全局化时，会导致不正确的对象记录被输出。

- Updates to RDOFF subdirectory, and changes to outrdf.c.

更新到 RDOFF 子目录，并更改到 outdf.c。

C.3.5 Version 0.93 released January 1997
C. 3.5 Version 0.93 发布于 1997 年 1 月

This release went out in a great hurry after semi−crippling bugs were found in 0.92.
在 0.92 中发现了半致残的错误之后，这个版本很快就发布了。

- Really *did* fix the stack overflows this time. *blush*

这次确实修复了堆栈溢出。 * 腮红 *

- Had problems with EA instruction sizes changing between passes, when an offset contained a forward reference and so 4 bytes were allocated for the offset in pass one; by pass two the symbol had been defined and happened to be a small absolute value, so only 1 byte got allocated, causing instruction size mismatch between passes and hence incorrect address calculations. Fixed.

当一个偏移量包含一个前向引用时，EA 指令的大小在传递之间发生变化，因此在第一个传递中为偏移量分配了 4 个字节; 在第二个传递中，符号被定义，碰巧是一个小的绝对值，因此只分配了 1 个字节，导致传递之间的指令大小不匹配，从而导致不正确的地址计算。修正。

- Stupid bug in the revised ELF section generation fixed (associated string−table section for .symtab was hard−coded as 7, even when this didn’t fit with the real section table). Was causing ‘ld’ to seg−fault under Linux.

愚蠢的错误在修订的 ELF 部分生成固定(相关的字符串表部分为。Symtab 被硬编码为 7，即使这不符合真正的 section 表)。在 Linux 下导致‘ ld’分隔错误。

- Included a new Borland C makefile, Makefile.bc2, donated by Fox Cutter <lmb@comtch.iea.com>.

包括一个新的 Borland c makefile，Makefile.bc2，由 Fox Cutter 捐赠。

C.3.6 Version 0.92 released January 1997
C. 3.6 版本 0.92,1997 年 1 月发布

- The FDIVP/FDIVRP and FSUBP/FSUBRP pairs had been inverted: this was fixed. This also affected the LCC driver.

FDIVP/FDIVRP 和 FSUBP/FSUBRP 对已经倒置: 这是固定的。这也影响 LCC 驱动程序。

- Fixed a bug regarding 32−bit effective addresses of the form `[other_register+ESP]`.

修正了一个关于 32 位有效地址表单[其他 _ 寄存器 + ESP]的错误。

- Documentary changes, notably documentation of the fact that Borland Win32 compilers use ‘obj’ rather than ‘win32’ object format.

文档更改，特别是 Borland win32 编译器使用‘ obj’而不是‘ Win32’对象格式的文档。

- Fixed the COMENT record in OBJ files, which was formatted incorrectly.

修正了 OBJ 文件中的 comment 记录格式不正确。

- Fixed a bug causing segfaults in large RDF files.

修正了一个导致大型 RDF 文件错误的错误。

- OBJ format now strips initial periods from segment and group definitions, in order to avoid complications with the local label syntax.

OBJ 格式现在从段和组定义中去除初始句点，以避免本地标签语法的复杂性。

- Fixed a bug in disassembling far calls and jumps in NDISASM.

修正了 NDISASM 中分解远程调用和跳转的错误。

- Added support for user−defined sections in COFF and ELF files.

增加了对 COFF 和 ELF 文件中用户定义部分的支持。

- Compiled the DOS binaries with a sensible amount of stack, to prevent stack overflows on any arithmetic expression containing parentheses.

使用合理数量的堆栈编译 DOS 二进制文件，以防止任何包含括号的算术表达式堆栈溢出。

- Fixed a bug in handling of files that do not terminate in a newline.

修正了在处理文件时不以换行结束的错误。

C.3.7 Version 0.91 released November 1996
C. 3.7 版本 0.91 发布于 1996 年 11 月

- Loads of bug fixes.

大量的 bug 修复。

- Support for RDF added.

支持 RDF。

- Support for DBG debugging format added.

增加了对 DBG 调试格式的支持。

- Support for 32−bit extensions to Microsoft OBJ format added.

支持微软 OBJ 格式的 32 位扩展。

- Revised for Borland C: some variable names changed, makefile added.

对 Borland c 进行了修改: 修改了一些变量名，添加了 makefile。

- LCC support revised to actually work.

LCC 支持修改为实际工作。

• JMP/CALL NEAR/FAR notation added.
添加了 JMP/CALL NEAR/FAR 符号。

• ‘a16’, ‘o16’, ‘a32’ and ‘o32’ prefixes added.
添加了前缀‘ a16’、‘ o16’、‘ a32’和‘ o32’。

• Range checking on short jumps implemented.
实施短距离跳跃距离检查。

• MMX instruction support added.
增加了 MMX 指令支持。

• Negative floating point constant support added.
增加了负浮点常数支持。

• Memory handling improved to bypass 64K barrier under DOS.
内存处理改进，在 DOS 下绕过 64k 屏障。

• `$` prefix to force treatment of reserved words as identifiers added.
$前缀强制将保留字处理为添加的标识符。

• Default−size mechanism for object formats added.
添加对象格式的默认大小机制。

• Compile−time configurability added.
增加了编译时可配置性。

• `#`, `@`, `~` and c{?} are now valid characters in labels.
,@，~ 和 c { ? }现在是标签中有效的字符。

• `-e` and `-k` options in NDISASM added.
在 NDISASM 中增加了 -e 和 -k 选项。

C.3.8 Version 0.90 released October 1996
C. 3.8 版本 0.90 于 1996 年 10 月发布

First release version. First support for object file output. Other changes from previous version (0.3x) too numerous to document.
First 发行版。 First 支持目标文件输出。其他更改(0.3 x)太多，无法记录。

Appendix D: Building NASM from Source
附录 d: 从源头构建 NASM

The source code for NASM is available from our website, http://wwww.nasm.us/, see section E.1.
NASM 的源代码可以从我们的网站 http://www.NASM.us/获得，见 e. 1 节。

D.1 Building from a Source Archive
D. 1 来自源文档的建筑

The source archives available on the web site should be capable of building on a number of platforms. This is the recommended method for building NASM to support platforms for which executables are not available.
网站上提供的源档案应能够建立在若干平台上。这是建立 NASM 的推荐方法，以支持那些没有可执行文件的平台。

On a system which has Unix shell (`sh`), run:
在一个有 Unix shell (sh)的系统上，运行:

```
sh configure
Sh 配置
make everything
创造一切
```

A number of options can be passed to `configure`; see `sh configure --help`.
可以传递许多选项来配置; 请参见 shconfigure-help。

A set of Makefiles for some other environments are also available; please see the file `Mkfiles/README`.
一组其他环境下的 Makefiles 也可以使用; 请参阅文件 Mkfiles/README。

To build the installer for the Windows platform, you will need the *Nullsoft Scriptable Installer*, NSIS, installed.
为了构建 Windows 平台的安装程序，您需要安装 Nullsoft Scriptable Installer，NSIS。

To build the documentation, you will need a set of additional tools. The documentation is not likely to be able to build on non−Unix systems.
为了构建文档，你需要一组额外的工具。这些文档不太可能在非 Unix 系统上构建。

D.2 Building from the `git` Repository
D. 2 从 git Repository 构建

The NASM development tree is kept in a source code repository using the `git` distributed source control system. The link is available on the website. This is recommended only to participate in the development of NASM or to assist with testing the development code.
NASM 开发树使用 git 分布式源代码控制系统保存在一个原始码储存库中。这个链接可以在网站上找到。建议仅用于参与 NASM 的开发或者协助测试开发代码。

To build NASM from the `git` repository you will need a Perl and, if building on a Unix system, GNU autoconf.
要从 git 存储库构建 NASM，您需要一个 Perl，如果构建在 Unix 系统上，则需要 GNU autoconf。

To build on a Unix system, run:
要在 Unix 系统上构建，运行:

```
sh autogen.sh
Sh autogen.sh
```

to create the `configure` script and then build as listed above.
创建配置脚本，然后按照上面列出的方式构建。

273

274

Appendix E: Contact Information
附录 e: 联络资料

E.1 Website
E. 1 网站

NASM has a website at `http://www.nasm.us/`.
NASM 的网站是 http://www.NASM.us/。

New releases, release candidates, and daily development snapshots of NASM are available from the official web site in source form as well as binaries for a number of common platforms.
NASM 的新版本、候选版本和每日开发快照可以从官方网站以源代码的形式获得，也可以从许多公共平台的二进制文件获得。

E.1.1 User Forums
E. 1.1 用户论坛

Users of NASM may find the Forums on the website useful. These are, however, not frequented much by the developers of NASM, so they are not suitable for reporting bugs.
NASM 的用户可能会发现网站上的论坛很有用。然而，NASM 的开发者并不经常使用这些论坛，所以它们不适合报告 bug。

E.1.2 Development Community
E. 1.2 发展共同体

The development of NASM is coordinated primarily though the `nasm-devel` mailing list. If you wish to participate in development of NASM, please join this mailing list. Subscription links and archives of past posts are available on the website.
NASM 的开发主要通过 NASM-devel 邮件列表进行协调。如果你想参与 NASM 的开发，请加入这个邮件列表。订阅链接和过去文章的档案可以在网站上找到。

E.2 Reporting Bugs
E. 2 报告错误

To report bugs in NASM, please use the bug tracker at `http://www.nasm.us/` (click on "Bug Tracker"), or if that fails then through one of the contacts in section E.1.
要报告 NASM 中的 Bug，请使用 http://www.NASM.us/(点击“ Bug Tracker”)的 Bug 跟踪器，或者如果失败，则通过 e. 1 节中的联系人之一进行报告。

Please read section 2.2 first, and don’t report the bug if it’s listed in there as a deliberate feature. (If you think the feature is badly thought out, feel free to send us reasons why you think it should be changed, but don’t just send us mail saying ‘This is a bug’ if the documentation says we did it on purpose.) Then read section 12.1, and don’t bother reporting the bug if it’s listed there.
请先阅读第 2.2 节，如果这个 bug 是故意列出来的，请不要报告。(如果你认为这个功能考虑得很糟糕，请随时给我们发送你认为应该更改它的理由，但是如果文档说我们是故意这样做的，不要只是给我们发送邮件说‘这是一个 bug’。)然后阅读第 12.1 节，如果列出了这个 bug，就不用麻烦去报告了。

If you do report a bug, *please* make sure your bug report includes the following information:
如果你确实报告了一个 bug，请确保你的 bug 报告包括以下信息:

• What operating system you’re running NASM under. Linux, FreeBSD, NetBSD, MacOS X, Win16, Win32, Win64, MS−DOS, OS/2, VMS, whatever.
您在哪个操作系统下运行 NASM。Linux，FreeBSD，NetBSD，MacOS x，Win16，Win32，Win64，MS-DOS，OS/2，VMS 等等。

- If you compiled your own executable from a source archive, compiled your own executable from `git`, used the standard distribution binaries from the website, or got an executable from somewhere else (e.g. a Linux distribution.) If you were using a locally built executable, try to reproduce the problem using one of the standard binaries, as this will make it easier for us to reproduce your problem prior to fixing it.

如果您从源文档编译自己的可执行文件，从 git 编译自己的可执行文件，使用网站上的标准发行版二进制文件，或者从其他地方获得一个可执行文件(例如 Linux 发行版)如果您使用的是本地构建的可执行文件，请尝试使用标准的二进制文件重现问题，因为这将使我们更容易在修复问题之前重现问题。

- Which version of NASM you're using, and exactly how you invoked it. Give us the precise command line, and the contents of the `NASMENV` environment variable if any.

您使用的是哪个版本的 NASM，以及您是如何调用它的。给我们精确的命令行，以及 NASMENV 环境变量的内容。

- Which versions of any supplementary programs you're using, and how you invoked them. If the problem only becomes visible at link time, tell us what linker you're using, what version of it you've got, and the exact linker command line. If the problem involves linking against object files generated by a compiler, tell us what compiler, what version, and what command line or options you used. (If you're compiling in an IDE, please try to reproduce the problem with the command−line version of the compiler.)

你正在使用的补充程序的版本，以及你如何调用它们。如果问题只在链接时才显现，请告诉我们您使用的是哪个链接器，它的版本是什么，以及确切的链接器命令行。如果问题涉及到对编译器生成的对象文件的链接，告诉我们什么编译器，什么版本，以及你使用了什么命令行或选项。(如果你正在 IDE 中编译，请尝试用编译器的命令行版本重现这个问题

- If at all possible, send us a NASM source file which exhibits the problem. If this causes copyright problems (e.g. you can only reproduce the bug in restricted−distribution code) then bear in mind the following two points: firstly, we guarantee that any source code sent to us for the purposes of debugging NASM will be used *only* for the purposes of debugging NASM, and that we will delete all our copies of it as soon as we have found and fixed the bug or bugs in question; and secondly, we would prefer *not* to be mailed large chunks of code anyway. The smaller the file, the better. A three−line sample file that does nothing useful *except* demonstrate the problem is much easier to

如果可能的话，请给我们发送一个显示问题的 NASM 源文件。如果这会导致版权问题(例如，你只能在限制发行的代码中复制 bug)，那么请记住以下两点: 第一，我们保证任何发送给我们用于调试 NASM 的源代码将仅用于调试 NASM，并且一旦我们发现并修复了有问题的 bug 或 bug，我们将删除我们所有的副本; 第二，我们宁愿不要邮寄大块的代码。文件越小越好。一个三行的示例文件，除了演示问题之外，没有任何有用的东西，要容易得多

275

work with than a fully fledged ten−thousand−line program. (Of course, some errors *do* only crop up in large files, so this may not be possible.)
而不是一个成熟的万行程序。(当然，有些错误只会在大文件中出现，所以这可能是不可能的

- A description of what the problem actually *is*. ‘It doesn’t work’ is *not* a helpful description! Please describe exactly what is happening that shouldn’t be, or what isn’t happening that should. Examples might be: ‘NASM generates an error message saying Line 3 for an error that’s actually on Line 5’; ‘NASM generates an error message that I believe it shouldn’t be generating at all’; ‘NASM fails to generate an error message that I believe it *should* be generating’; ‘the object file produced from this source code crashes my linker’; ‘the ninth byte of the output file is 66 and I think it should be 77 instead’.

关于问题实际是什么的描述。“它不工作”不是一个有用的描述！请准确描述发生了什么不应该发生的事情，或者发生了什么不应该发生的事情。例子可能是: ‘ NASM 生成一个错误消息，说实际上在第 5 行的错误在第 3 行’; ‘ NASM 生成一个我认为根本不应该生成的错误消息’; ‘ NASM 未能生成一个我认为应该生成的错误消息’; ‘从这个源代码生成的目标文件崩溃了我的链接器’; ‘输出文件的第九个字节是 66，我认为应该是 77’。

- If you believe the output file from NASM to be faulty, send it to us. That allows us to determine whether our own copy of NASM generates the same file, or whether the problem is related to portability issues between our development platforms and yours. We can handle binary files mailed to us as MIME attachments, uuencoded, and even BinHex. Alternatively, we may be able to provide an FTP site you can upload the suspect files to; but mailing them is easier for us.

如果您认为 NASM 的输出文件有问题，请发送给我们。这使我们能够确定我们自己的 NASM 副本是否生成相同的文件，或者这个问题是否与我们的开发平台和您的平台之间的可移植性问题有关。我们可以处理作为 MIME 附件、 uuencoded 甚至 BinHex 邮寄给我们的二进制文件。或者，我们可以提供一个 FTP 站点，你可以上传可疑文件到这个站点; 但是邮寄它们对我们来说更容易。

- Any other information or data files that might be helpful. If, for example, the problem involves NASM failing to generate an object file while TASM can generate an equivalent file without trouble, then send us *both* object files, so we can see what TASM is doing differently from us.

任何其他可能有用的信息或数据文件。例如，如果问题涉及 NASM 无法生成目标文件，而 TASM 可以毫无困难地生成等效文件，那么就向我们发送两个目标文件，这样我们就可以看到 TASM 与我们做的不同之处。

276
276

Index
索引

277

278

279

280

281

282

283

284