



x86 Assembly Language Reference Manual

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-5477-10
January 2005

Copyright 2005 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2005 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



040910@9495



Contents

Preface	7
1 Overview of the Solaris x86 Assembler	11
Assembler Overview	11
Syntax Differences Between x86 Assemblers	12
Assembler Command Line	12
2 Solaris x86 Assembly Language Syntax	13
Lexical Conventions	13
Statements	13
Tokens	15
Instructions, Operands, and Addressing	17
Instructions	17
Operands	18
Assembler Directives	20
3 Instruction Set Mapping	25
Instruction Overview	25
General-Purpose Instructions	26
Data Transfer Instructions	26
Binary Arithmetic Instructions	30
Decimal Arithmetic Instructions	31
Logical Instructions	32
Shift and Rotate Instructions	32
Bit and Byte Instructions	33

Control Transfer Instructions	35
String Instructions	38
I/O Instructions	39
Flag Control (EFLAG) Instructions	40
Segment Register Instructions	41
Miscellaneous Instructions	41
Floating-Point Instructions	42
Data Transfer Instructions (Floating Point)	42
Basic Arithmetic Instructions (Floating-Point)	43
Comparison Instructions (Floating-Point)	45
Transcendental Instructions (Floating-Point)	46
Load Constants (Floating-Point) Instructions	47
Control Instructions (Floating-Point)	47
SIMD State Management Instructions	49
MMX Instructions	49
Data Transfer Instructions (MMX)	50
Conversion Instructions (MMX)	50
Packed Arithmetic Instructions (MMX)	51
Comparison Instructions (MMX)	52
Logical Instructions (MMX)	53
Shift and Rotate Instructions (MMX)	53
State Management Instructions (MMX)	54
SSE Instructions	54
SIMD Single-Precision Floating-Point Instructions (SSE)	55
MXCSR State Management Instructions (SSE)	61
64-Bit SIMD Integer Instructions (SSE)	61
Miscellaneous Instructions (SSE)	62
SSE2 Instructions	63
SSE2 Packed and Scalar Double-Precision Floating-Point Instructions	63
SSE2 Packed Single-Precision Floating-Point Instructions	70
SSE2 128-Bit SIMD Integer Instructions	70
SSE2 Miscellaneous Instructions	72
Operating System Support Instructions	73
64-Bit AMD Opteron Considerations	75

Index 77

Tables

TABLE 3-1	Data Transfer Instructions	26
TABLE 3-2	Binary Arithmetic Instructions	30
TABLE 3-3	Decimal Arithmetic Instructions	32
TABLE 3-4	Logical Instructions	32
TABLE 3-5	Shift and Rotate Instructions	33
TABLE 3-6	Bit and Byte Instructions	34
TABLE 3-7	Control Transfer Instructions	36
TABLE 3-8	String Instructions	38
TABLE 3-9	I/O Instructions	40
TABLE 3-10	Flag Control Instructions	40
TABLE 3-11	Segment Register Instructions	41
TABLE 3-12	Miscellaneous Instructions	42
TABLE 3-13	Data Transfer Instructions (Floating-Point)	42
TABLE 3-14	Basic Arithmetic Instructions (Floating-Point)	44
TABLE 3-15	Comparison Instructions (Floating-Point)	45
TABLE 3-16	Transcendental Instructions (Floating-Point)	46
TABLE 3-17	Load Constants Instructions (Floating-Point)	47
TABLE 3-18	Control Instructions (Floating-Point)	47
TABLE 3-19	SIMD State Management Instructions	49
TABLE 3-20	Data Transfer Instructions (MMX)	50
TABLE 3-21	Conversion Instructions (MMX)	50
TABLE 3-22	Packed Arithmetic Instructions (MMX)	51
TABLE 3-23	Comparison Instructions (MMX)	52
TABLE 3-24	Logical Instructions (MMX)	53
TABLE 3-25	Shift and Rotate Instructions (MMX)	53
TABLE 3-26	State Management Instructions (MMX)	54
TABLE 3-27	Data Transfer Instructions (SSE)	55

TABLE 3-28	Packed Arithmetic Instructions (SSE)	56
TABLE 3-29	Comparison Instructions (SSE)	58
TABLE 3-30	Logical Instructions (SSE)	59
TABLE 3-31	Shuffle and Unpack Instructions (SSE)	59
TABLE 3-32	Conversion Instructions (SSE)	60
TABLE 3-33	MXCSR State Management Instructions (SSE)	61
TABLE 3-34	64-Bit SIMD Integer Instructions (SSE)	61
TABLE 3-35	Miscellaneous Instructions (SSE)	62
TABLE 3-36	SSE2 Data Movement Instructions	64
TABLE 3-37	SSE2 Packed Arithmetic Instructions	65
TABLE 3-38	SSE2 Logical Instructions	66
TABLE 3-39	SSE2 Compare Instructions	67
TABLE 3-40	SSE2 Shuffle and Unpack Instructions	67
TABLE 3-41	SSE2 Conversion Instructions	68
TABLE 3-42	SSE2 Packed Single-Precision Floating-Point Instructions	70
TABLE 3-43	SSE2 128-Bit SIMD Integer Instructions	71
TABLE 3-44	SSE2 Miscellaneous Instructions	72
TABLE 3-45	Operating System Support Instructions	73

Preface

The *x86 Assembly Language Reference Manual* documents the syntax of the Solaris™ x86 assembly language. This manual is provided to help experienced programmers understand the assembly language output of Solaris compilers. This manual is neither an introductory book about assembly language programming nor a reference manual for the x86 architecture.

Note – In this document the term “x86” refers to 64-bit and 32-bit systems manufactured using processors compatible with the AMD64 or Intel Xeon/Pentium product families. For supported systems, see the *Solaris 10 Hardware Compatibility List*.

Who Should Use This Book

This manual is intended for experienced x86 assembly language programmers who are familiar with the x86 architecture.

Before You Read This Book

You should have a thorough knowledge of assembly language programming in general and be familiar with the x86 architecture in specific. You should be familiar with the ELF object file format. This manual assumes that you have the following documentation available for reference:

- *IA-32 Intel Architecture Software Developer's Manual* (Intel Corporation, 2004).
Volume 1: *Basic Architecture*. Volume 2: *Instruction Set Reference A-M*. Volume 3: *Instruction Set Reference N-Z*. Volume 4: *System Programming Guide*.

- *AMD64 Architecture Programmer's Manual* (Advanced Micro Devices, 2003). Volume 1: *Application Programming*. Volume 2: *System Programming*. Volume 3: *General-Purpose and System Instructions*. Volume 4: *128-Bit Media Instructions*. Volume 5: *64-Bit Media and x87 Floating-Point Instructions*.
- *Linker and Libraries Guide*
- *Sun Studio 9: C User's Guide*
- *Sun Studio 9: Fortran User's Guide* and *Fortran Programming Guide*
- Man pages for the `as(1)`, `ld(1)`, and `dis(1)` utilities.

How This Book Is Organized

[Chapter 1](#) provides an overview of the x86 functionality supported by the Solaris x86 assembler.

[Chapter 2](#) documents the syntax of the Solaris x86 assembly language.

[Chapter 3](#) maps Solaris x86 assembly language instruction mnemonics to the native x86 instruction set.

Accessing Sun Documentation Online

The docs.sun.comSM Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print. For a list of documents and how to order them, see “Buy printed documentation” at `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	The command to remove a file is <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . Perform a <i>patch analysis</i> . Do <i>not</i> save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	<code>machine_name%</code>
C shell superuser prompt	<code>machine_name#</code>
Bourne shell and Korn shell prompt	<code>\$</code>

TABLE P-2 Shell Prompts *(Continued)*

Shell	Prompt
Bourne shell and Korn shell superuser prompt	#

Overview of the Solaris x86 Assembler

This chapter provides a brief overview of the Solaris x86 assembler `as`. This chapter discusses the following topics:

- “Assembler Overview” on page 11
- “Syntax Differences Between x86 Assemblers” on page 12
- “Assembler Command Line” on page 12

Assembler Overview

The Solaris x86 assembler `as` translates Solaris x86 assembly language into Executable and Linking Format (ELF) relocatable object files that can be linked with other object files to create an executable file or a shared object file. (See Chapter 7, “Object File Format,” in *Linker and Libraries Guide*, for a complete discussion of ELF object file format.) The assembler supports macro processing by the C preprocessor (`cpp`) or the `m4` macro processor. The assembler supports the instruction sets of the following CPUs:

Intel 8086/8088 processors
Intel 286 processor
Intel386 processor
Intel486 processor
Intel Pentium processor
Intel Pentium Pro processor
Intel Pentium II processor
Pentium II Xeon processor
Intel Celeron processor
Intel Pentium III processor
Pentium III Xeon processor
Advanced Micro Devices Athlon processor

Syntax Differences Between x86 Assemblers

There is no standard assembly language for the x86 architecture. Vendor implementations of assemblers for the x86 architecture instruction sets differ in syntax and functionality. The syntax of the Solaris x86 assembler is compatible with the syntax of the assembler distributed with earlier releases of the UNIX operating system (this syntax is sometimes termed “AT&T syntax”). Developers familiar with other assemblers derived from the original UNIX assemblers, such as the Free Software Foundation’s `gas`, will find the syntax of the Solaris x86 assembler very straightforward.

However, the syntax of x86 assemblers distributed by Intel and Microsoft (sometimes termed “Intel syntax”) differs significantly from the syntax of the Solaris x86 assembler. These differences are most pronounced in the handling of instruction operands:

- The Solaris and Intel assemblers use the opposite order for source and destination operands.
- The Solaris assembler specifies the size of memory operands by adding a suffix to the instruction mnemonic, while the Intel assembler prefixes the memory operands.
- The Solaris assembler prefixes immediate operands with a dollar sign (\$) (ASCII 0x24), while the Intel assembler does not delimit immediate operands.

See [Chapter 2](#) for additional differences between x86 assemblers.

Assembler Command Line

During the translation of higher-level languages such as C and Fortran, the compilers might invoke `as` using the alias `fbe` (“Fortran back end”). You can invoke the assembler manually from the shell command line with either name, `as` or `fbe`. See the `as(1)` man page for the definitive discussion of command syntax and command line options.

Solaris x86 Assembly Language Syntax

This chapter documents the syntax of the Solaris x86 assembly language.

- “Lexical Conventions” on page 13
- “Instructions, Operands, and Addressing” on page 17
- “Assembler Directives” on page 20

Lexical Conventions

This section discusses the lexical conventions of the Solaris x86 assembly language.

Statements

An x86 assembly language program consists of one or more files containing *statements*. A *statement* consists of *tokens* separated by *whitespace* and terminated by either a newline character (ASCII 0x0A) or a semicolon (;) (ASCII 0x3B). *Whitespace* consists of spaces (ASCII 0x20), tabs (ASCII 0x09), and formfeeds (ASCII 0x0B) that are not contained in a string or comment. More than one statement can be placed on a single input line provided that each statement is terminated by a semicolon. A statement can consist of a *comment*. *Empty statements*, consisting only of whitespace, are allowed.

Comments

A *comment* can be appended to a statement. The comment consists of the slash character (/) (ASCII 0x2F) followed by the text of the comment. The comment is terminated by the newline that terminates the statement.

Labels

A *label* can be placed at the beginning of a statement. During assembly, the label is assigned the current value of the active location counter and serves as an instruction operand. There are two types of labels: *symbolic* and *numeric*.

Symbolic Labels

A *symbolic* label consists of an *identifier* (or *symbol*) followed by a colon (:) (ASCII 0x3A). Symbolic labels must be defined only once. Symbolic labels have *global* scope and appear in the object file's symbol table.

Symbolic labels with identifiers beginning with a period (.) (ASCII 0x2E) are considered to have *local* scope and are not included in the object file's symbol table.

Numeric Labels

A *numeric* label consists of a single digit in the range zero (0) through nine (9) followed by a colon (:). Numeric labels are used only for local reference and are not included in the object file's symbol table. Numeric labels have limited scope and can be redefined repeatedly.

When a numeric label is used as a reference (as an instruction operand, for example), the suffixes *b* ("backward") or *f* ("forward") should be added to the numeric label. For numeric label *N*, the reference *Nb* refers to the nearest label *N* defined *before* the reference, and the reference *Nf* refers to the nearest label *N* defined *after* the reference. The following example illustrates the use of numeric labels:

```
1:          / define numeric label "1"
one:       / define symbolic label "one"

/ ... assembler code ...

jmp  1f    / jump to first numeric label "1" defined
          / after this instruction
          / (this reference is equivalent to label "two")

jmp  1b    / jump to last numeric label "1" defined
          / before this instruction
          / (this reference is equivalent to label "one")

1:        / redefine label "1"
two:     / define symbolic label "two"

jmp  1b    / jump to last numeric label "1" defined
          / before this instruction
          / (this reference is equivalent to label "two")
```

Tokens

There are five classes of tokens:

- Identifiers (symbols)
- Keywords
- Numerical constants
- String Constants
- Operators

Identifiers

An *identifier* is an arbitrarily-long sequence of letters and digits. The first character must be a letter; the underscore (`_`) (ASCII 0x5F) and the period (`.`) (ASCII 0x2E) are considered to be letters. Case is significant: uppercase and lowercase letters are different.

Keywords

Keywords such as x86 instruction mnemonics (“opcodes”) and assembler directives are reserved for the assembler and should not be used as identifiers. See [Chapter 3](#) for a list of the Solaris x86 mnemonics. See “[Assembler Directives](#)” on page 20 for the list of as assembler directives.

Numerical Constants

Numbers in the x86 architecture can be *integers* or *floating point*. Integers can be *signed* or *unsigned*, with signed integers represented in two’s complement representation. Floating-point numbers can be: single-precision floating-point; double-precision floating-point; and double-extended precision floating-point.

Integer Constants

Integers can be expressed in several bases:

- **Decimal.** Decimal integers begin with a non-zero digit followed by zero or more decimal digits (0–9).
- **Binary.** Binary integers begin with “0b” or “0B” followed by zero or more binary digits (0, 1).
- **Octal.** Octal integers begin with zero (0) followed by zero or more octal digits (0–7).
- **Hexadecimal.** Hexadecimal integers begin with “0x” or “0X” followed by one or more hexadecimal digits (0–9, A–F). Hexadecimal digits can be either uppercase or lowercase.

Floating Point Constants

Floating point constants have the following format:

- **Sign** (optional) – either plus (+) or minus (–)
- **Integer** (optional) – zero or more decimal digits (0–9)
- **Fraction** (optional) – decimal point (.) followed by zero or more decimal digits
- **Exponent** (optional) – the letter “e” or “E”, followed by an optional sign (plus or minus), followed by one or more decimal digits (0–9)

A valid floating point constant must have either an integer part or a fractional part.

String Constants

A *string* constant consists of a sequence of characters enclosed in double quotes (") (ASCII 0x22). To include a double-quote character ("), single-quote character ('), or backslash character (\) within a string, precede the character with a backslash (\) (ASCII 0x5C). A character can be expressed in a string as its ASCII value in octal preceded by a backslash (for example, the letter “J” could be expressed as “\112”). The assembler accepts the following escape sequences in strings:

Escape Sequence	Character Name	ASCII Value (hex)
\n	newline	0A
\r	carriage return	0D
\b	backspace	08
\t	horizontal tab	09
\f	form feed	0C
\v	vertical tab	0B

Operators

The assembler supports the following operators for use in expressions. Operators have no assigned precedence. Expressions can be grouped in square brackets ([]) to establish precedence.

- + Addition
- Subtraction
- * Multiplication
- \/ Division
- & Bitwise logical AND

	Bitwise logical OR
>>	Shift right
<<	Shift left
\%	Remainder
!	Bitwise logical AND NOT
^	Bitwise logical XOR

Note – The asterisk (*), slash (/), and percent sign (%) characters are overloaded. When used as operators in an expression, these characters must be preceded by the backslash character (\).

Instructions, Operands, and Addressing

Instructions are operations performed by the CPU. *Operands* are entities operated upon by the instruction. *Addresses* are the locations in memory of specified data.

Instructions

An *instruction* is a statement that is executed at runtime. An x86 instruction statement can consist of four parts:

- Label (optional)
- Instruction (required)
- Operands (instruction specific)
- Comment (optional)

See “Statements” on page 13 for the description of labels and comments.

The terms *instruction* and *mnemonic* are used interchangeably in this document to refer to the names of x86 instructions. Although the term *opcode* is sometimes used as a synonym for *instruction*, this document reserves the term *opcode* for the hexadecimal representation of the instruction value.

For most instructions, the Solaris x86 assembler mnemonics are the same as the Intel or AMD mnemonics. However, the Solaris x86 mnemonics might appear to be different because the Solaris mnemonics are suffixed with a one-character modifier that specifies the size of the instruction operands. That is, the Solaris assembler derives its operand type information from the instruction name and the suffix. If a mnemonic is specified with no type suffix, the operand type defaults to `long`. Possible operand types and their instruction suffixes are:

- b Byte (8-bit)
- w Word (16-bit)
- l Long (32-bit) (default)
- q Quadword (64-bit)

The assembler recognizes the following suffixes for x87 floating-point instructions:

- [no suffix] Instruction operands are registers only
- l ("long") Instruction operands are 64-bit
- s ("short") Instruction operands are 32-bit

See [Chapter 3](#) for a mapping between Solaris x86 assembly language mnemonics and the equivalent Intel or AMD mnemonics.

Operands

An x86 instruction can have zero to three operands. Operands are separated by commas (,) (ASCII 0x2C). For instructions with two operands, the first (lefthand) operand is the *source* operand, and the second (righthand) operand is the *destination* operand (that is, *source*→*destination*).

Note – The Intel assembler uses the opposite order (*destination*←*source*) for operands.

Operands can be *immediate* (that is, constant expressions that evaluate to an inline value), *register* (a value in the processor number registers), or *memory* (a value stored in memory). An *indirect* operand contains the address of the actual operand value. Indirect operands are specified by prefixing the operand with an asterisk (*) (ASCII 0x2A). Only jump and call instructions can use indirect operands.

- *Immediate* operands are prefixed with a dollar sign (\$) (ASCII 0x24)
- *Register* names are prefixed with a percent sign (%) (ASCII 0x25)

- *Memory* operands are specified either by the name of a variable or by a register that contains the address of a variable. A variable name implies the address of a variable and instructs the computer to reference the contents of memory at that address. Memory references have the following syntax:
segment : offset (base, index, scale) .
 - *Segment* is any of the x86 architecture segment registers. *Segment* is optional: if specified, it must be separated from *offset* by a colon (:). If *segment* is omitted, the value of %ds (the default segment register) is assumed.
 - *Offset* is the displacement from *segment* of the desired memory value. *Offset* is optional.
 - *Base* and *index* can be any of the general 32-bit number registers.
 - *Scale* is a factor by which *index* is to be multiplied before being added to *base* to specify the address of the operand. *Scale* can have the value of 1, 2, 4, or 8. If *scale* is not specified, the default value is 1.

Some examples of memory addresses are:

```
movl var, %eax
```

Move the contents of memory location *var* into number register %eax.

```
movl %cs:var, %eax
```

Move the contents of memory location *var* in the code segment (register %cs) into number register %eax.

```
movl $var, %eax
```

Move the address of *var* into number register %eax.

```
movl array_base(%esi), %eax
```

Add the address of memory location *array_base* to the contents of number register %esi to determine an address in memory. Move the contents of this address into number register %eax.

```
movl (%ebx, %esi, 4), %eax
```

Multiply the contents of number register %esi by 4 and add the result to the contents of number register %ebx to produce a memory reference. Move the contents of this memory location into number register %eax.

```
movl struct_base(%ebx, %esi, 4), %eax
```

Multiply the contents of number register %esi by 4, add the result to the contents of number register %ebx, and add the result to the address of *struct_base* to produce an address. Move the contents of this address into number register %eax.

Assembler Directives

Directives are commands that are part of the assembler syntax but are not related to the x86 processor instruction set. All assembler directives begin with a period (.) (ASCII 0x2E).

`.align integer, pad`

The `.align` directive causes the next data generated to be aligned modulo *integer* bytes. *Integer* must be a positive integer expression and must be a power of 2. If specified, *pad* is an integer byte value used for padding. The default value of *pad* for the `text` section is 0x90 (nop); for other sections, the default value of *pad* is zero (0).

`.ascii "string"`

The `.ascii` directive places the characters in *string* into the object module at the current location but does *not* terminate the string with a null byte (\0). *String* must be enclosed in double quotes (") (ASCII 0x22). The `.ascii` directive is not valid for the `.bss` section.

`.bcd integer`

The `.bcd` directive generates a packed decimal (80-bit) value into the current section. The `.bcd` directive is not valid for the `.bss` section.

`.bss`

The `.bss` directive changes the current section to `.bss`.

`.bss symbol, integer`

Define *symbol* in the `.bss` section and add *integer* bytes to the value of the location counter for `.bss`. When issued with arguments, the `.bss` directive does not change the current section to `.bss`. *Integer* must be positive.

`.byte byte1, byte2, ..., byteN`

The `.byte` directive generates initialized bytes into the current section. The `.byte` directive is not valid for the `.bss` section. Each *byte* must be an 8-bit value.

`.2byte expression1, expression2, ..., expressionN`

Refer to the description of the `.value` directive.

`.4byte expression1, expression2, ..., expressionN`

Refer to the description of the `.long` directive.

`.8byte expression1, expression2, ..., expressionN`

Refer to the description of the `.quad` directive.

`.comm name, size, alignment`

The `.comm` directive allocates storage in the data section. The storage is referenced by the identifier *name*. *Size* is measured in bytes and must be a positive integer. *Name* cannot be predefined. *Alignment* is optional. If *alignment* is specified, the address of *name* is aligned to a multiple of *alignment*.

`.data`

The `.data` directive changes the current section to `.data`.

- .double *float*
The .double directive generates a double-precision floating-point constant into the current section. The .double directive is not valid for the .bss section.
- .even
The .even directive aligns the current program counter (.) to an even boundary.
- .ext *expression1, expression2, ..., expressionN*
The .ext directive generates an 80387 80-bit floating point constant for each *expression* into the current section. The .ext directive is not valid for the .bss section.
- .file "*string*"
The .file directive creates a symbol table entry where *string* is the symbol name and STT_FILE is the symbol table type. *String* specifies the name of the source file associated with the object file.
- .float *float*
The .float directive generates a single-precision floating-point constant into the current section. The .float directive is not valid in the .bss section.
- .globl *symbol1, symbol2, ..., symbolN*
The .globl directive declares each *symbol* in the list to be *global*. Each symbol is either defined externally or defined in the input file and accessible in other files. Default bindings for the symbol are overridden. A global symbol definition in one file satisfies an undefined reference to the same global symbol in another file. Multiple definitions of a defined global symbol are not allowed. If a defined global symbol has more than one definition, an error occurs. The .globl directive only declares the symbol to be global in scope, it does not define the symbol.
- .group *group, section, #comdat*
The .group directive adds *section* to a COMDAT *group*. Refer to "COMDAT Section" in *Linker and Libraries Guide* for additional information about COMDAT.
- .hidden *symbol1, symbol2, ..., symbolN*
The .hidden directive declares each *symbol* in the list to have *hidden* linker scoping. All references to *symbol* within a dynamic module bind to the definition within that module. *Symbol* is not visible outside of the module.
- .ident "*string*"
The .ident directive creates an entry in the .comment section containing *string*. *String* is any sequence of characters, not including the double quote ("). To include the double quote character within a string, precede the double quote character with a backslash (\) (ASCII 0x5C).
- .lcomm *name, size, alignment*
The .lcomm directive allocates storage in the .bss section. The storage is referenced by the symbol *name*, and has a size of *size* bytes. *Name* cannot be predefined, and *size* must be a positive integer. If *alignment* is specified, the address of *name* is aligned to a multiple of *alignment* bytes. If *alignment* is not specified, the default alignment is 4 bytes.

- .local *symbol1, symbol2, ..., symbolN*
 The .local directive declares each *symbol* in the list to be *local*. Each symbol is defined in the input file and not accessible to other files. Default bindings for the symbols are overridden. Symbols declared with the .local directive take precedence over *weak* and *global* symbols. (See “Symbol Table Section” in *Linker and Libraries Guide* for a description of global and weak symbols.) Because local symbols are not accessible to other files, local symbols of the same name may exist in multiple files. The .local directive only declares the symbol to be local in scope, it does not define the symbol.
- .long *expression1, expression2, ..., expressionN*
 The .long directive generates a long integer (32-bit, two’s complement value) for each *expression* into the current section. Each *expression* must be a 32-bit value and must evaluate to an integer value. The .long directive is not valid for the .bss section.
- .popsection
 The .popsection directive pops the top of the section stack and continues processing of the popped section.
- .previous
 The .previous directive continues processing of the previous section.
- .pushsection *section*
 The .pushsection directive pushes the specified section onto the section stack and switches to another section.
- .quad *expression1, expression2, ..., expressionN*
 The .quad directive generates an initialized word (64-bit, two’s complement value) for each *expression* into the current section. Each *expression* must be a 64-bit value, and must evaluate to an integer value. The .quad directive is not valid for the .bss section.
- .rel *symbol@ type*
 The .rel directive generates the specified relocation entry *type* for the specified *symbol*. The .lit directive supports TLS (thread-local storage). Refer to Chapter 8, “Thread-Local Storage,” in *Linker and Libraries Guide* for additional information about TLS.
- .section *section, attributes*
 The .section directive makes *section* the current section. If *section* does not exist, a new section with the specified name and attributes is created. If *section* is a non-reserved section, *attributes* must be included the first time *section* is specified by the .section directive.
- .set *symbol, expression*
 The .set directive assigns the value of *expression* to *symbol*. *Expression* can be any legal expression that evaluates to a numerical value.
- .skip *integer, value*
 While generating values for any data section, the .skip directive causes *integer* bytes to be skipped over, or, optionally, filled with the specified *value*.

- `.sleb128 expression`
The `.sleb128` directive generates a signed, little-endian, base 128 number from *expression*.
- `.string "string"`
The `.string` directive places the characters in *string* into the object module at the current location and terminates the string with a null byte (`\0`). *String* must be enclosed in double quotes (`"`) (ASCII 0x22). The `.string` directive is not valid for the `.bss` section.
- `.symbolic symbol1, symbol2, ..., symbolN`
The `.symbolic` directive declares each *symbol* in the list to have *symbolic* linker scoping. All references to *symbol* within a dynamic module bind to the definition within that module. Outside of the module, *symbol* is treated as global.
- `.tbss`
The `.tbss` directive changes the current section to `.tbss`. The `.tbss` section contains uninitialized TLS data objects that will be initialized to zero by the runtime linker.
- `.tcomm`
The `.tcomm` directive defines a TLS common block.
- `.tdata`
The `.tdata` directive changes the current section to `.tdata`. The `.tdata` section contains the initialization image for initialized TLS data objects.
- `.text`
The `.text` directive defines the current section as `.text`.
- `.uleb128 expression`
The `.uleb128` directive generates an unsigned, little-endian, base 128 number from *expression*.
- `.value expression1, expression2, ..., expressionN`
The `.value` directive generates an initialized word (16-bit, two's complement value) for each *expression* into the current section. Each *expression* must be a 16-bit integer value. The `.value` directive is not valid for the `.bss` section.
- `.weak symbol1, symbol2, ..., symbolN`
The `.weak` directive declares each *symbol* in the argument list to be defined either externally or in the input file and accessible to other files. Default bindings of the symbol are overridden by the `.weak` directive. A *weak* symbol definition in one file satisfies an undefined reference to a global symbol of the same name in another file. Unresolved *weak* symbols have a default value of zero. The link editor does not resolve these symbols. If a *weak* symbol has the same name as a defined *global* symbol, the weak symbol is ignored and no error results. The `.weak` directive does not define the symbol.
- `.zero expression`
While filling a data section, the `.zero` directive fills the number of bytes specified by *expression* with zero (0).

Instruction Set Mapping

This chapter provides a general mapping between the Solaris x86 assembly language mnemonics and the Intel or Advanced Micro Devices (AMD) mnemonics.

- “Instruction Overview” on page 25
- “General-Purpose Instructions” on page 26
- “Floating-Point Instructions” on page 42
- “SIMD State Management Instructions” on page 49
- “MMX Instructions” on page 49
- “SSE Instructions” on page 54
- “SSE2 Instructions” on page 63
- “Operating System Support Instructions” on page 73
- “64-Bit AMD Opteron Considerations” on page 75

Instruction Overview

It is beyond the scope of this manual to document the x86 architecture instruction set. This chapter provides a general mapping between the Solaris x86 assembly language mnemonics and the Intel or AMD mnemonics to enable you to refer to your vendor’s documentation for detailed information about a specific instruction. Instructions are grouped by functionality in tables with the following sections:

- Solaris mnemonic
- Intel/AMD mnemonic
- Description (short)
- Notes

For certain Solaris mnemonics, the allowed data type suffixes for that mnemonic are indicated in braces ({}), following the mnemonic. For example, `bswap{1q}` indicates that the following mnemonics are valid: `bswap`, `bswap1` (which is the default and equivalent to `bswap`), and `bswapq`. See “Instructions” on page 17 for information on data type suffixes.

To locate a specific Solaris x86 mnemonic, look up the mnemonic in the index.

General-Purpose Instructions

The general-purpose instructions perform basic data movement, memory addressing, arithmetic and logical operations, program flow control, input/output, and string operations on integer, pointer, and BCD data types.

Data Transfer Instructions

The data transfer instructions move data between memory and the general-purpose and segment registers, and perform operations such as conditional moves, stack access, and data conversion.

TABLE 3-1 Data Transfer Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>bswap{lq}</code>	BSWAP	byte swap	<code>bswapq</code> valid only under -xarch=amd64
<code>cbtw</code>	CBW	convert byte to word	
<code>cld</code>	CDQ	convert doubleword to quadword	<code>%eax</code> → <code>%edx:%eax</code>
<code>cltq</code>	CDQE	convert doubleword to quadword	<code>%eax</code> → <code>%rax</code> <code>cltq</code> valid only under -xarch=amd64
<code>cmova{wlq}, cmov{wlq}.a</code>	CMOVA	conditional move if above	<code>cmovaq</code> valid only under -xarch=amd64
<code>cmovae{wlq}, cmov{wlq}.ae</code>	CMOVAE	conditional move if above or equal	<code>cmovaeq</code> valid only under -xarch=amd64
<code>cmovb{wlq}, cmov{wlq}.b</code>	CMOVB	conditional move if below	<code>cmovbq</code> valid only under -xarch=amd64

TABLE 3-1 Data Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cmovbe{wlq}, cmov{wlq}.be	CMOVBE	conditional move if below or equal	cmovbeq valid only under -xarch=amd64
cmovc{wlq}, cmov{wlq}.c	CMOVC	conditional move if carry	cmovcq valid only under -xarch=amd64
cmove{wlq}, cmov{wlq}.e	CMOVE	conditional move if equal	cmoveq valid only under -xarch=amd64
cmovg{wlq}, cmov{wlq}.g	CMOVG	conditional move if greater	cmovgq valid only under -xarch=amd64
cmovge{wlq}, cmov{wlq}.ge	CMOVGE	conditional move if greater or equal	cmovgeq valid only under -xarch=amd64
cmovl{wlq}, cmov{wlq}.l	CMOVL	conditional move if less	cmovlq valid only under -xarch=amd64
cmovle{wlq}, cmov{wlq}.le	CMOVLE	conditional move if less or equal	cmovleq valid only under -xarch=amd64
cmovna{wlq}, cmov{wlq}.na	CMOVNA	conditional move if not above	cmovnaq valid only under -xarch=amd64
cmovnae{wlq}, cmov{wlq}.nae	CMOVNAE	conditional move if not above or equal	cmovnaeq valid only under -xarch=amd64
cmovnb{wlq}, cmov{wlq}.nb	CMOVNB	conditional move if not below	cmovnbq valid only under -xarch=amd64
cmovnbe{wlq}, cmov{wlq}.nbe	CMOVNBE	conditional move if not below or equal	cmovnbeq valid only under -xarch=amd64
cmovnc{wlq}, cmov{wlq}.nc	CMOVNC	conditional move if not carry	cmovncq valid only under -xarch=amd64
cmovne{wlq}, cmov{wlq}.ne	CMOVNE	conditional move if not equal	cmovneq valid only under -xarch=amd64

TABLE 3-1 Data Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>cmovng{wlq}, cmov{wlq}.ng</code>	CMOVNG	conditional move if greater	<code>cmovngq</code> valid only under -xarch=amd64
<code>cmovnge{wlq}, cmov{wlq}.nge</code>	CMOVNGE	conditional move if not greater or equal	<code>cmovngeq</code> valid only under -xarch=amd64
<code>cmovnl{wlq}, cmov{wlq}.nl</code>	CMOVNL	conditional move if not less	<code>cmovnlq</code> valid only under -xarch=amd64
<code>cmovnle{wlq}, cmov{wlq}.nle</code>	CMOVNLE	conditional move if not above or equal	<code>cmovnleq</code> valid only under -xarch=amd64
<code>cmovno{wlq}, cmov{wlq}.no</code>	CMOVNO	conditional move if not overflow	<code>cmovnoq</code> valid only under -xarch=amd64
<code>cmovnp{wlq}, cmov{wlq}.np</code>	CMOVNP	conditional move if not parity	<code>cmovnpq</code> valid only under -xarch=amd64
<code>cmovns{wlq}, cmov{wlq}.ns</code>	CMOVNS	conditional move if not sign (non-negative)	<code>cmovnsq</code> valid only under -xarch=amd64
<code>cmovnz{wlq}, cmov{wlq}.nz</code>	CMOVNZ	conditional move if not zero	<code>cmovnzq</code> valid only under -xarch=amd64
<code>cmovo{wlq}, cmov{wlq}.o</code>	CMOVO	conditional move if overflow	<code>cmovoq</code> valid only under -xarch=amd64
<code>cmovp{wlq}, cmov{wlq}.p</code>	CMOVPP	conditional move if parity	<code>cmovpq</code> valid only under -xarch=amd64
<code>cmovpe{wlq}, cmov{wlq}.pe</code>	CMOVPE	conditional move if parity even	<code>cmovpeq</code> valid only under -xarch=amd64
<code>cmovpo{wlq}, cmov{wlq}.po</code>	CMOVPO	conditional move if parity odd	<code>cmovpoq</code> valid only under -xarch=amd64
<code>cmovs{wlq}, cmov{wlq}.s</code>	CMOVSS	conditional move if sign (negative)	<code>cmovsq</code> valid only under -xarch=amd64

TABLE 3-1 Data Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>cmovz{wlq}, cmov{wlq}.z</code>	CMOVZ	conditional move if zero	<code>cmovzq</code> valid only under -xarch=amd64
<code>cmpxchg{bwlq}</code>	CMPXCHG	compare and exchange	<code>cmpxchgq</code> valid only under -xarch=amd64
<code>cmpxchg8b</code>	CMPXCHG8B	compare and exchange 8 bytes	
<code>cqtd</code>	CQO	convert quadword to octword	<code>%rax → %rdx:%rax</code> <code>cqtd</code> valid only under -xarch=amd64
<code>cqto</code>	CQO	convert quadword to octword	<code>%rax → %rdx:%rax</code> <code>cqto</code> valid only under -xarch=amd64
<code>cwtd</code>	CWD	convert word to doubleword	<code>%ax → %dx:%ax</code>
<code>cwtl</code>	CWDE	convert word to doubleword in <code>%eax</code> register	<code>%ax → %eax</code>
<code>mov{bwlq}</code>	MOV	move data between immediate values, general purpose registers, segment registers, and memory	<code>movq</code> valid only under -xarch=amd64
<code>movabs{bwlq}</code>	MOVABS	move immediate value to register	<code>movabs</code> valid only under -xarch=amd64
<code>movabs{bwlq}A</code>	MOVABS	move immediate value to register {AL, AX, GAX, RAX}	<code>movabs</code> valid only under -xarch=amd64
<code>movsb{wlq}, movsw{lq}</code>	MOVSB	move and sign extend	<code>movsbq</code> and <code>movswq</code> valid only under -xarch=amd64
<code>movzb{wlq}, movzw{lq}</code>	MOVZB	move and zero extend	<code>movzbq</code> and <code>movzwq</code> valid only under -xarch=amd64

TABLE 3-1 Data Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
pop{wlq}	POP	pop stack	popq valid only under -xarch=amd64
popaw	POPA	pop general-purpose registers from stack	popaw invalid under -xarch=amd64
popal, popa	POPAD	pop general-purpose registers from stack	invalid under -xarch=amd64
push{wlq}	PUSH	push onto stack	pushq valid only under -xarch=amd64
pushaw	PUSHA	push general-purpose registers onto stack	pushaw invalid under -xarch=amd64
pushal, pusha	PUSHAD	push general-purpose registers onto stack	invalid under -xarch=amd64
xadd{bwlq}	XADD	exchange and add	xaddq valid only under -xarch=amd64
xchg{bwlq}	XCHG	exchange	xchgq valid only under -xarch=amd64
xchg{bwlq}A	XCHG	exchange	xchgqA valid only under -xarch=amd64

Binary Arithmetic Instructions

The binary arithmetic instructions perform basic integer computations on operands in memory or the general-purpose registers.

TABLE 3-2 Binary Arithmetic Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
adc{bwlq}	ADC	add with carry	adcq valid only under -xarch=amd64
add{bwlq}	ADD	integer add	addq valid only under -xarch=amd64

TABLE 3-2 Binary Arithmetic Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cmp{bwlq}	CMP	compare	cmpq valid only under -xarch=amd64
dec{bwlq}	DEC	decrement	decq valid only under -xarch=amd64
div{bwlq}	DIV	divide (unsigned)	divq valid only under -xarch=amd64
idiv{bwlq}	IDIV	divide (signed)	idivq valid only under -xarch=amd64
imul{bwlq}	IMUL	multiply (signed)	imulq valid only under -xarch=amd64
inc{bwlq}	INC	increment	incq valid only under -xarch=amd64
mul{bwlq}	MUL	multiply (unsigned)	mulq valid only under -xarch=amd64
neg{bwlq}	NEG	negate	negq valid only under -xarch=amd64
sbb{bwlq}	SBB	subtract with borrow	sbbq valid only under -xarch=amd64
sub{bwlq}	SUB	subtract	subq valid only under -xarch=amd64

Decimal Arithmetic Instructions

The decimal arithmetic instructions perform decimal arithmetic on binary coded decimal (BCD) data.

TABLE 3-3 Decimal Arithmetic Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
aaa	AAA	ASCII adjust after addition	invalid under -xarch=amd64
aad	AAD	ASCII adjust before division	invalid under -xarch=amd64
aam	AAM	ASCII adjust after multiplication	invalid under -xarch=amd64
aas	AAS	ASCII adjust after subtraction	invalid under -xarch=amd64
daa	DAA	decimal adjust after addition	invalid under -xarch=amd64
das	DAS	decimal adjust after subtraction	invalid under -xarch=amd64

Logical Instructions

The logical instructions perform basic logical operations on their operands.

TABLE 3-4 Logical Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
and{bwlq}	AND	bitwise logical AND	andq valid only under -xarch=amd64
not{bwlq}	NOT	bitwise logical NOT	notq valid only under -xarch=amd64
or{bwlq}	OR	bitwise logical OR	orq valid only under -xarch=amd64
xor{bwlq}	XOR	bitwise logical exclusive OR	xorq valid only under -xarch=amd64

Shift and Rotate Instructions

The shift and rotate instructions shift and rotate the bits in their operands.

TABLE 3-5 Shift and Rotate Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
rcl{bwlq}	RCL	rotate through carry left	rclq valid only under -xarch=amd64
rcr{bwlq}	RCR	rotate through carry right	rcrq valid only under -xarch=amd64
rol{bwlq}	ROL	rotate left	rolq valid only under -xarch=amd64
ror{bwlq}	ROR	rotate right	rorq valid only under -xarch=amd64
sal{bwlq}	SAL	shift arithmetic left	salq valid only under -xarch=amd64
sar{bwlq}	SAR	shift arithmetic right	sarq valid only under -xarch=amd64
shl{bwlq}	SHL	shift logical left	shlq valid only under -xarch=amd64
shld{bwlq}	SHLD	shift left double	shldq valid only under -xarch=amd64
shr{bwlq}	SHR	shift logical right	shrq valid only under -xarch=amd64
shrd{bwlq}	SHRD	shift right double	shrdq valid only under -xarch=amd64

Bit and Byte Instructions

The bit instructions test and modify individual bits in operands. The byte instructions set the value of a byte operand to indicate the status of flags in the `%eflags` register.

TABLE 3-6 Bit and Byte Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
bsf{wlq}	BSF	bit scan forward	bsfq valid only under -xarch=amd64
bsr{wlq}	BSR	bit scan reverse	bsrq valid only under -xarch=amd64
bt{wlq}	BT	bit test	btq valid only under -xarch=amd64
btc{wlq}	BTC	bit test and complement	btcq valid only under -xarch=amd64
btr{wlq}	BTR	bit test and reset	btrq valid only under -xarch=amd64
bts{wlq}	BTS	bit test and set	btsq valid only under -xarch=amd64
seta	SETA	set byte if above	
setae	SETAE	set byte if above or equal	
setb	SETB	set byte if below	
setbe	SETBE	set byte if below or equal	
setc	SETC	set byte if carry	
sete	SETE	set byte if equal	
setg	SETG	set byte if greater	
setge	SETGE	set byte if greater or equal	
setl	SETL	set byte if less	
setle	SETLE	set byte if less or equal	
setna	SETNA	set byte if not above	
setnae	SETNAE	set byte if not above or equal	
setnb	SETNB	set byte if not below	

TABLE 3-6 Bit and Byte Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
setnbe	SETNBE	set byte if not below or equal	
setnc	SETNC	set byte if not carry	
setne	SETNE	set byte if not equal	
setng	SETNG	set byte if not greater	
setnge	SETNGE	set byte if not greater or equal	
setnl	SETNL	set byte if not less	
setnle	SETNLE	set byte if not less or equal	
setno	SETNO	set byte if not overflow	
setnp	SETNP	set byte if not parity	
setns	SETNS	set byte if not sign (non-negative)	
setnz	SETNZ	set byte if not zero	
seto	SETO	set byte if overflow	
setp	SETP	set byte if parity	
setpe	SETPE	set byte if parity even	
setpo	SETPO	set byte if parity odd	
sets	SETS	set byte if sign (negative)	
setz	SETZ	set byte if zero	
test{bwlq}	TEST	logical compare	testq valid only under -xarch=amd64

Control Transfer Instructions

The control transfer instructions control the flow of program execution.

TABLE 3-7 Control Transfer Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
bound{w1}	BOUND	detect value out of range	boundw invalid under -xarch=amd64
call	CALL	call procedure	
enter	ENTER	high-level procedure entry	
int	INT	software interrupt	
into	INTO	interrupt on overflow	invalid under -xarch=amd64
iret	IRET	return from interrupt	
ja	JA	jump if above	
jae	JAE	jump if above or equal	
jb	JB	jump if below	
jbe	JBE	jump if below or equal	
jc	JC	jump if carry	
jcxz	JCXZ	jump register %cx zero	
je	JE	jump if equal	
jecxz	JECXZ	jump register %ecx zero	invalid under -xarch=amd64
jg	JG	jump if greater	
jge	JGE	jump if greater or equal	
jl	JL	jump if less	
jle	JLE	jump if less or equal	
jmp	JMP	jump	
jnae	JNAE	jump if not above or equal	
jnb	JNB	jump if not below	
jnbе	JNBE	jump if not below or equal	
jnc	JNC	jump if not carry	

TABLE 3-7 Control Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
jne	JNE	jump if not equal	
jng	JNG	jump if not greater	
jnge	JNGE	jump if not greater or equal	
jnl	JNL	jump if not less	
jnle	JNLE	jump if not less or equal	
jno	JNO	jump if not overflow	
jnp	JNP	jump if not parity	
jns	JNS	jump if not sign (non-negative)	
jnz	JNZ	jump if not zero	
jo	JO	jump if overflow	
jp	JP	jump if parity	
jpe	JPE	jump if parity even	
jpo	JPO	jump if parity odd	
js	JS	jump if sign (negative)	
jz	JZ	jump if zero	
lcall	CALL	call far procedure	valid as indirect only for -xarg=amd64
leave	LEAVE	high-level procedure exit	
loop	LOOP	loop with %ecx counter	
loope	LOOPE	loop with %ecx and equal	
loopne	LOOPNE	loop with %ecx and not equal	
loopnz	LOOPNZ	loop with %ecx and not zero	
loopz	LOOPZ	loop with %ecx and zero	

TABLE 3-7 Control Transfer Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
lret	RET	return from far procedure	valid as indirect only for -xarch=amd64
ret	RET	return	

String Instructions

The string instructions operate on strings of bytes. Operations include storing strings in memory, loading strings from memory, comparing strings, and scanning strings for substrings.

Note – The Solaris mnemonics for certain instructions differ slightly from the Intel/AMD mnemonics. Alphabetization of the table below is by the Solaris mnemonic. All string operations default to long (doubleword).

TABLE 3-8 String Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cmps {q}	CMPS	compare string	cmpsq valid only under -xarch=amd64
cmpsb	CMPSB	compare byte string	
cmpsl	CMPSD	compare doubleword string	
cmpsw	CMPSW	compare word string	
lods {q}	LODS	load string	lodsq valid only under -xarch=amd64
lods b	LODSB	load byte string	
lods l	LODSD	load doubleword string	
lods w	LODSW	load word string	
movs {q}	MOVS	move string	movsq valid only under -xarch=amd64

TABLE 3-8 String Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movsb	MOVSB	move byte string	movsb is not movsb{wlq}. See Table 3-1
movsl, smovl	MOVSD	move doubleword string	
movsw, smovw	MOVSW	move word string	movsw is not movsw{lq}. See Table 3-1
rep	REP	repeat while %ecx not zero	
repnz	REPNE	repeat while not equal	
repnz	REPZ	repeat while not zero	
repz	REPE	repeat while equal	
repz	REPZ	repeat while zero	
scas{q}	SCAS	scan string	scasq valid only under -xarch=amd64
scasb	SCASB	scan byte string	
scasl	SCASD	scan doubleword string	
scasw	SCASW	scan word string	
stos{q}	STOS	store string	stosq valid only under -xarch=amd64
stosb	STOSB	store byte string	
stosl	STOSD	store doubleword string	
stosw	STOSW	store word string	

I/O Instructions

The input/output instructions transfer data between the processor's I/O ports, registers, and memory.

TABLE 3-9 I/O Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
in	IN	read from a port	
ins	INS	input string from a port	
insb	INSB	input byte string from port	
insl	INSD	input doubleword string from port	
insw	INSW	input word string from port	
out	OUT	write to a port	
outs	OUTS	output string to port	
outsb	OUTSB	output byte string to port	
outsl	OUTSD	output doubleword string to port	
outsw	OUTSW	output word string to port	

Flag Control (EFLAG) Instructions

The status flag control instructions operate on the bits in the `%eflags` register.

TABLE 3-10 Flag Control Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
clc	CLC	set carry flag	
cld	CLD	clear direction flag	
cli	CLI	clear interrupt flag	
cmc	CMC	complement carry flag	
lahf	LAHF	load flags into <code>%ah</code> register	
popfw	POPF	pop <code>%eflags</code> from stack	

TABLE 3–10 Flag Control Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
popf{lq}	POPFL	pop %eflags from stack	popfq valid only under -xarch=amd64
pushfw	PUSHF	push %eflags onto stack	
pushf{lq}	PUSHFL	push %eflags onto stack	pushfq valid only under -xarch=amd64
sahf	SAHF	store %ah register into flags	
stc	STC	set carry flag	
std	STD	set direction flag	
sti	STI	set interrupt flag	

Segment Register Instructions

The segment register instructions load far pointers (segment addresses) into the segment registers.

TABLE 3–11 Segment Register Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
lds{wl}	LDS	load far pointer using %ds	lds1 and ldsw invalid under -xarch=amd64
les{wl}	LES	load far pointer using %es	les1 and lesw invalid under -xarch=amd64
lfs{wl}	LFS	load far pointer using %fs	
lgs{wl}	LGS	load far pointer using %gs	
lss{wl}	LSS	load far pointer using %ss	

Miscellaneous Instructions

The instructions documented in this section provide a number of useful functions.

TABLE 3–12 Miscellaneous Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cpuid	CPUID	processor identification	
lea{wlq}	LEA	load effective address	leaq valid only under -xarch=amd64
nop	NOP	no operation	
ud2	UD2	undefined instruction	
xlat	XLAT	table lookup translation	
xlatb	XLATB	table lookup translation	

Floating-Point Instructions

The floating point instructions operate on floating-point, integer, and binary coded decimal (BCD) operands.

Data Transfer Instructions (Floating Point)

The data transfer instructions move floating-point, integer, and BCD values between memory and the floating point registers.

TABLE 3–13 Data Transfer Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fbld	FBLD	load BCD	
fbstp	FBSTP	store BCD and pop	
fcmovb	FCMOVB	floating-point conditional move if below	
fcmovbe	FCMOVBE	floating-point conditional move if below or equal	

TABLE 3-13 Data Transfer Instructions (Floating-Point) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>fcmove</code>	FCMOVE	floating-point conditional move if equal	
<code>fcmovnb</code>	FCMOVNB	floating-point conditional move if not below	
<code>fcmovnbe</code>	FCMOVNBE	floating-point conditional move if not below or equal	
<code>fcmovne</code>	FCMOVNE	floating-point conditional move if not equal	
<code>fcmovnu</code>	FCMOVNU	floating-point conditional move if unordered	
<code>fcmovu</code>	FCMOVU	floating-point conditional move if unordered	
<code>fild</code>	FILD	load integer	
<code>fist</code>	FIST	store integer	
<code>fistp</code>	FISTP	store integer and pop	
<code>fld</code>	FLD	load floating-point value	
<code>fst</code>	FST	store floating-point value	
<code>fstp</code>	FSTP	store floating-point value and pop	
<code>fxch</code>	FXCH	exchange registers	

Basic Arithmetic Instructions (Floating-Point)

The basic arithmetic instructions perform basic arithmetic operations on floating-point and integer operands.

TABLE 3-14 Basic Arithmetic Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fabs	FABS	absolute value	
fadd	FADD	add floating-point	
faddp	FADDP	add floating-point and pop	
fchs	FCHS	change sign	
fdiv	FDIV	divide floating-point	
fdivp	FDIVP	divide floating-point and pop	
fdivr	FDIVR	divide floating-point reverse	
fdivrp	FDIVRP	divide floating-point reverse and pop	
fiadd	FIADD	add integer	
fdiv	FIDIV	divide integer	
fdivr	FIDIVR	divide integer reverse	
fimul	FIMUL	multiply integer	
fisub	FISUB	subtract integer	
fisubr	FISUBR	subtract integer reverse	
fmul	FMUL	multiply floating-point	
fmulp	FMULP	multiply floating-point and pop	
fprem	FPREM	partial remainder	
fprem1	FPREM1	IEEE partial remainder	
frndint	FRNDINT	round to integer	
fscale	FSCALE	scale by power of two	
fsqrt	FSQRT	square root	
fsub	FSUB	subtract floating-point	

TABLE 3–14 Basic Arithmetic Instructions (Floating-Point) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fsubp	FSUBP	subtract floating-point and pop	
fsubr	FSUBR	subtract floating-point reverse	
fsubrp	FSUBRP	subtract floating-point reverse and pop	
fextract	EXTRACT	extract exponent and significand	

Comparison Instructions (Floating-Point)

The floating-point comparison instructions operate on floating-point or integer operands.

TABLE 3–15 Comparison Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fcom	FCOM	compare floating-point	
fcomi	FCOMI	compare floating-point and set %eflags	
fcomip	FCOMIP	compare floating-point, set %eflags, and pop	
fcomp	FCOMP	compare floating-point and pop	
fcompp	FCOMPP	compare floating-point and pop twice	
ficom	FICOM	compare integer	
ficompl	FICOMPL	compare integer and pop	
ftst	FTST	test floating-point (compare with 0.0)	

TABLE 3-15 Comparison Instructions (Floating-Point) (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fucom	FUCOM	unordered compare floating-point	
fucomi	FUCOMI	unordered compare floating-point and set %eflags	
fucomip	FUCOMIP	unordered compare floating-point, set %eflags, and pop	
fucomp	FUCOMP	unordered compare floating-point and pop	
fucompp	FUCOMPP	compare floating-point and pop twice	
fxam	FXAM	examine floating-point	

Transcendental Instructions (Floating-Point)

The transcendental instructions perform trigonometric and logarithmic operations on floating-point operands.

TABLE 3-16 Transcendental Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
f2xm1	F2XM1	computes $2^x - 1$	
fcos	FCOS	cosine	
fpatan	FPATAN	partial arctangent	
fptan	FPTAN	partial tangent	
fsin	FSIN	sine	
fsincos	FSINCOS	sine and cosine	
fyl2x	FYL2X	computes $y * \log_2 x$	
fyl2xp1	FYL2XP1	computes $y * \log_2(x+1)$	

Load Constants (Floating-Point) Instructions

The load constants instructions load common constants, such as π , into the floating-point registers.

TABLE 3-17 Load Constants Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
f1d1	FLD1	load +1.0	
f1d12e	FLDL2E	load $\log_2 e$	
f1d12t	FLDL2T	load $\log_2 10$	
f1d1g2	FLDLG2	load $\log_{10} 2$	
f1dln2	FLDLN2	load $\log_e 2$	
f1dpi	FLDPI	load π	
f1dz	FLDZ	load +0.0	

Control Instructions (Floating-Point)

The floating-point control instructions operate on the floating-point register stack and save and restore the floating-point state.

TABLE 3-18 Control Instructions (Floating-Point)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
fclex	FCLEX	clear floating-point exception flags after checking for error conditions	
fdecstp	FDECSTP	decrement floating-point register stack pointer	
ffree	FFREE	free floating-point register	
fincstp	FINCSTP	increment floating-point register stack pointer	

TABLE 3-18 Control Instructions (Floating-Point) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>finit</code>	<code>FINIT</code>	initialize floating-point unit after checking error conditions	
<code>fldcw</code>	<code>FLDCW</code>	load floating-point unit control word	
<code>fldenv</code>	<code>FLDENV</code>	load floating-point unit environment	
<code>fnclx</code>	<code>FNCLEX</code>	clear floating-point exception flags without checking for error conditions	
<code>fninit</code>	<code>FNINIT</code>	initialize floating-point unit without checking error conditions	
<code>fnop</code>	<code>FNOP</code>	floating-point no operation	
<code>fnsave</code>	<code>FNSAVE</code>	save floating-point unit state without checking error conditions	
<code>fnstcw</code>	<code>FNSTCW</code>	store floating-point unit control word without checking error conditions	
<code>fnstenv</code>	<code>FNSTENV</code>	store floating-point unit environment without checking error conditions	
<code>fnstsw</code>	<code>FNSTSW</code>	store floating-point unit status word without checking error conditions	
<code>frstor</code>	<code>FRSTOR</code>	restore floating-point unit state	
<code>fsave</code>	<code>FSAVE</code>	save floating-point unit state after checking error conditions	

TABLE 3–18 Control Instructions (Floating-Point) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>fstcw</code>	FSTCW	store floating-point unit control word after checking error conditions	
<code>fstenv</code>	FSTENV	store floating-point unit environment after checking error conditions	
<code>fstsw</code>	FSTSW	store floating-point unit status word after checking error conditions	
<code>fwait</code>	FWAIT	wait for floating-point unit	
<code>wait</code>	WAIT	wait for floating-point unit	

SIMD State Management Instructions

The `fxsave` and `fxrstor` instructions save and restore the state of the floating-point unit and the MMX, XMM, and MXCSR registers.

TABLE 3–19 SIMD State Management Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>fxrstor</code>	FXRSTOR	restore floating-point unit and SIMD state	
<code>fxsave</code>	FXSAVE	save floating-point unit and SIMD state	

MMX Instructions

The MMX instructions enable x86 processors to perform single-instruction, multiple-data(SIMD) operations on packed byte, word, doubleword, or quadword integer operands contained in memory, in MMX registers, or in general-purpose registers.

Data Transfer Instructions (MMX)

The data transfer instructions move doubleword and quadword operands between MMX registers and between MMX registers and memory.

TABLE 3–20 Data Transfer Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movd	MOVD	move doubleword	movdq valid only under -xarch=amd64
movq	MOVQ	move quadword	valid only under -xarch=amd64

Conversion Instructions (MMX)

The conversion instructions pack and unpack bytes, words, and doublewords.

TABLE 3–21 Conversion Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
packssdw	PACKSSDW	pack doublewords into words with signed saturation	
packsswb	PACKSSWB	pack words into bytes with signed saturation	
packuswb	PACKUSWB	pack words into bytes with unsigned saturation	
punpckhbw	PUNPCKHBW	unpack high-order bytes	
punpckhdq	PUNPCKHDQ	unpack high-order doublewords	
punpckhwd	PUNPCKHWD	unpack high-order words	
punpcklbw	PUNPCKLBW	unpack low-order bytes	
punpckldq	PUNPCKLDQ	unpack low-order doublewords	

TABLE 3–21 Conversion Instructions (MMX) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
punpcklwd	PUNPCKLWD	unpack low-order words	

Packed Arithmetic Instructions (MMX)

The packed arithmetic instructions perform packed integer arithmetic on packed byte, word, and doubleword integers.

TABLE 3–22 Packed Arithmetic Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
paddb	PADDB	add packed byte integers	
padd	PADD	add packed doubleword integers	
paddsb	PADDSB	add packed signed byte integers with signed saturation	
paddsw	PADDSW	add packed signed word integers with signed saturation	
paddusb	PADDUSB	add packed unsigned byte integers with unsigned saturation	
paddusw	PADDUSW	add packed unsigned word integers with unsigned saturation	
paddw	PADDW	add packed word integers	
pmaddwd	PMADDWD	multiply and add packed word integers	
pmulhw	PMULHW	multiply packed signed word integers and store high result	
pmullw	PMULLW	multiply packed signed word integers and store low result	

TABLE 3–22 Packed Arithmetic Instructions (MMX) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>psubb</code>	<code>PSUBB</code>	subtract packed byte integers	
<code>psubd</code>	<code>PSUBD</code>	subtract packed doubleword integers	
<code>psubsb</code>	<code>PSUBSB</code>	subtract packed signed byte integers with signed saturation	
<code>psubsw</code>	<code>PSUBSW</code>	subtract packed signed word integers with signed saturation	
<code>psubusb</code>	<code>PSUBUSB</code>	subtract packed unsigned byte integers with unsigned saturation	
<code>psubusw</code>	<code>PSUBUSW</code>	subtract packed unsigned word integers with unsigned saturation	
<code>psubw</code>	<code>PSUBW</code>	subtract packed word integers	

Comparison Instructions (MMX)

The compare instructions compare packed bytes, words, or doublewords.

TABLE 3–23 Comparison Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>pcmpeqb</code>	<code>PCMPEQB</code>	compare packed bytes for equal	
<code>pcmpeqd</code>	<code>PCMPEQD</code>	compare packed doublewords for equal	
<code>pcmpeqw</code>	<code>PCMPEQW</code>	compare packed words for equal	
<code>pcmpgtb</code>	<code>PCMPGTB</code>	compare packed signed byte integers for greater than	

TABLE 3–23 Comparison Instructions (MMX) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
pcmpgtd	PCMPGTD	compare packed signed doubleword integers for greater than	
pcmpgtw	PCMPGTW	compare packed signed word integers for greater than	

Logical Instructions (MMX)

The logical instructions perform logical operations on quadword operands.

TABLE 3–24 Logical Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
pand	PAND	bitwise logical AND	
pandn	PANDN	bitwise logical AND NOT	
por	POR	bitwise logical OR	
pxor	PXOR	bitwise logical XOR	

Shift and Rotate Instructions (MMX)

The shift and rotate instructions operate on packed bytes, words, doublewords, or quadwords in 64-bit operands.

TABLE 3–25 Shift and Rotate Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
pslld	PSLLD	shift packed doublewords left logical	
psllq	PSLLQ	shift packed quadword left logical	
psllw	PSLLW	shift packed words left logical	

TABLE 3–25 Shift and Rotate Instructions (MMX) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
psrad	PSRAD	shift packed doublewords right arithmetic	
psraw	PSRAW	shift packed words right arithmetic	
psrld	PSRLD	shift packed doublewords right logical	
psrlq	PSRLQ	shift packed quadword right logical	
psrlw	PSRLW	shift packed words right logical	

State Management Instructions (MMX)

The `emms` (EMMS) instruction clears the MMX state from the MMX registers.

TABLE 3–26 State Management Instructions (MMX)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
emms	EMMS	empty MMX state	

SSE Instructions

SSE instructions are an extension of the SIMD execution model introduced with the MMX technology. SSE instructions are divided into four subgroups:

- SIMD single-precision floating-point instructions that operate on the XMM registers
- MXSCR state management instructions
- 64-bit SIMD integer instructions that operate on the MMX registers
- Instructions that provide cache control, prefetch, and instruction ordering functionality

SIMD Single-Precision Floating-Point Instructions (SSE)

The SSE SIMD instructions operate on packed and scalar single-precision floating-point values located in the XMM registers or memory.

Data Transfer Instructions (SSE)

The SSE data transfer instructions move packed and scalar single-precision floating-point operands between XMM registers and between XMM registers and memory.

TABLE 3-27 Data Transfer Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movaps	MOVAPS	move four aligned packed single-precision floating-point values between XMM registers or memory	
movhyps	MOVHPS	move two packed single-precision floating-point values from the high quadword of an XMM register to the low quadword of another XMM register	
movhyps	MOVHPS	move two packed single-precision floating-point values to or from the high quadword of an XMM register or memory	
movlhyps	MOVLHPS	move two packed single-precision floating-point values from the low quadword of an XMM register to the high quadword of another XMM register	

TABLE 3–27 Data Transfer Instructions (SSE) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movlps	MOVLPS	move two packed single-precision floating-point values to or from the low quadword of an XMM register or memory	
movmskps	MOVMSKPS	extract sign mask from four packed single-precision floating-point values	
movss	MOVSS	move scalar single-precision floating-point value between XMM registers or memory	
movups	MOVUPS	move four unaligned packed single-precision floating-point values between XMM registers or memory	

Packed Arithmetic Instructions (SSE)

SSE packed arithmetic instructions perform packed and scalar arithmetic operations on packed and scalar single-precision floating-point operands.

TABLE 3–28 Packed Arithmetic Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
addps	ADDPS	add packed single-precision floating-point values	
addss	ADDSS	add scalar single-precision floating-point values	
divps	DIVPS	divide packed single-precision floating-point values	

TABLE 3–28 Packed Arithmetic Instructions (SSE) (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
divss	DIVSS	divide scalar single-precision floating-point values	
maxps	MAXPS	return maximum packed single-precision floating-point values	
maxss	MAXSS	return maximum scalar single-precision floating-point values	
minps	MINPS	return minimum packed single-precision floating-point values	
minss	MINSS	return minimum scalar single-precision floating-point values.	
mulps	MULPS	multiply packed single-precision floating-point values	
mulss	MULSS	multiply scalar single-precision floating-point values	
rcpps	RCPPS	compute reciprocals of packed single-precision floating-point values	
rcpss	RCPSS	compute reciprocal of scalar single-precision floating-point values	
rsqrtps	RSQRTPS	compute reciprocals of square roots of packed single-precision floating-point values	
rsqrtss	RSQRTSS	compute reciprocal of square root of scalar single-precision floating-point values	

TABLE 3–28 Packed Arithmetic Instructions (SSE) (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>sqrtps</code>	<code>SQRTPS</code>	compute square roots of packed single-precision floating-point values	
<code>sqrtss</code>	<code>SQRTSS</code>	compute square root of scalar single-precision floating-point values	
<code>subps</code>	<code>SUBPS</code>	subtract packed single-precision floating-point values	
<code>subss</code>	<code>SUBSS</code>	subtract scalar single-precision floating-point values	

Comparison Instructions (SSE)

The SSE compare instructions compare packed and scalar single-precision floating-point operands.

TABLE 3–29 Comparison Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>cmppps</code>	<code>CMPPS</code>	compare packed single-precision floating-point values	
<code>cmpss</code>	<code>CMPS</code>	compare scalar single-precision floating-point values	
<code>comiss</code>	<code>COMISS</code>	perform ordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register	
<code>ucomiss</code>	<code>UCOMISS</code>	perform unordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register	

Logical Instructions (SSE)

The SSE logical instructions perform bitwise AND, AND NOT, OR, and XOR operations on packed single-precision floating-point operands.

TABLE 3–30 Logical Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
andnps	ANDNPS	perform bitwise logical AND NOT of packed single-precision floating-point values	
andps	ANDPS	perform bitwise logical AND of packed single-precision floating-point values	
orps	ORPS	perform bitwise logical OR of packed single-precision floating-point values	
xorps	XORPS	perform bitwise logical XOR of packed single-precision floating-point values	

Shuffle and Unpack Instructions (SSE)

The SSE shuffle and unpack instructions shuffle or interleave single-precision floating-point values in packed single-precision floating-point operands.

TABLE 3–31 Shuffle and Unpack Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
shufps	SHUFPS	shuffles values in packed single-precision floating-point operands	

TABLE 3–31 Shuffle and Unpack Instructions (SSE) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
unpckhps	UNPCKHPS	unpacks and interleaves the two high-order values from two single-precision floating-point operands	
unpcklps	UNPCKLPS	unpacks and interleaves the two low-order values from two single-precision floating-point operands	

Conversion Instructions (SSE)

The SSE conversion instructions convert packed and individual doubleword integers into packed and scalar single-precision floating-point values.

TABLE 3–32 Conversion Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cvtpi2ps	CVTPI2PS	convert packed doubleword integers to packed single-precision floating-point values	
cvtps2pi	CVTPS2PI	convert packed single-precision floating-point values to packed doubleword integers	
cvtssi2ss	CVTSS2SS	convert doubleword integer to scalar single-precision floating-point value	
cvtss2si	CVTSS2SI	convert scalar single-precision floating-point value to a doubleword integer	

TABLE 3–32 Conversion Instructions (SSE) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>cvttps2pi</code>	<code>CVTTPS2PI</code>	convert with truncation packed single-precision floating-point values to packed doubleword integers	
<code>cvttss2si</code>	<code>CVTTSS2SI</code>	convert with truncation scalar single-precision floating-point value to scalar doubleword integer	

MXCSR State Management Instructions (SSE)

The MXCSR state management instructions save and restore the state of the MXCSR control and status register.

TABLE 3–33 MXCSR State Management Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>ldmxcsr</code>	<code>LDMXCSR</code>	load <code>%mxcsr</code> register	
<code>stmcsr</code>	<code>STMXCSR</code>	save <code>%mxcsr</code> register state	

64–Bit SIMD Integer Instructions (SSE)

The SSE 64-bit SIMD integer instructions perform operations on packed bytes, words, or doublewords in MMX registers.

TABLE 3–34 64–Bit SIMD Integer Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>pavgb</code>	<code>PAVGB</code>	compute average of packed unsigned byte integers	
<code>pavgw</code>	<code>PAVGW</code>	compute average of packed unsigned word integers	

TABLE 3-34 64-Bit SIMD Integer Instructions (SSE) (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
pextrw	PEXTRW	extract word	
pinsrw	PINSRW	insert word	
pmaxsw	PMAXSW	maximum of packed signed word integers	
pmaxub	PMAXUB	maximum of packed unsigned byte integers	
pminsw	PMINSW	minimum of packed signed word integers	
pminub	PMINUB	minimum of packed unsigned byte integers	
pmovmskb	PMOVMASKB	move byte mask	
pmulhbw	PMULHUBW	multiply packed unsigned integers and store high result	
psadbw	PSADBW	compute sum of absolute differences	
pshufw	PSHUFW	shuffle packed integer word in MMX register	

Miscellaneous Instructions (SSE)

The following instructions control caching, prefetching, and instruction ordering.

TABLE 3-35 Miscellaneous Instructions (SSE)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
maskmovq	MASKMOVQ	non-temporal store of selected bytes from an MMX register into memory	
movntps	MOVNTPS	non-temporal store of four packed single-precision floating-point values from an XMM register into memory	

TABLE 3-35 Miscellaneous Instructions (SSE) *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movntq	MOVNTQ	non-temporal store of quadword from an MMX register into memory	
prefetchnta	PREFETCHNTA	prefetch data into non-temporal cache structure and into a location close to the processor	
prefetcht0	PREFETCHT0	prefetch data into all levels of the cache hierarchy	
prefetcht1	PREFETCHT1	prefetch data into level 2 cache and higher	
prefetcht2	PREFETCHT2	prefetch data into level 2 cache and higher	
sfence	SFENCE	serialize store operations	

SSE2 Instructions

SSE2 instructions are an extension of the SIMD execution model introduced with the MMX technology and the SSE extensions. SSE2 instructions are divided into four subgroups:

- Packed and scalar double-precision floating-point instructions
- Packed single-precision floating-point conversion instructions
- 128-bit SIMD integer instructions
- Instructions that provide cache control and instruction ordering functionality

SSE2 Packed and Scalar Double-Precision Floating-Point Instructions

The SSE2 packed and scalar double-precision floating-point instructions operate on double-precision floating-point operands.

SSE2 Data Movement Instructions

The SSE2 data movement instructions move double-precision floating-point data between XMM registers and memory.

TABLE 3-36 SSE2 Data Movement Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movapd	MOVAPD	move two aligned packed double-precision floating-point values between XMM registers and memory	
movhpd	MOVHPD	move high packed double-precision floating-point value to or from the high quadword of an XMM register and memory	
movlpd	MOVLPD	move low packed single-precision floating-point value to or from the low quadword of an XMM register and memory	
movmskpd	MOVMSKPD	extract sign mask from two packed double-precision floating-point values	
movsd	MOVSD	move scalar double-precision floating-point value between XMM registers and memory.	
movupd	MOVUPD	move two unaligned packed double-precision floating-point values between XMM registers and memory	

SSE2 Packed Arithmetic Instructions

The SSE2 arithmetic instructions operate on packed and scalar double-precision floating-point operands.

TABLE 3–37 SSE2 Packed Arithmetic Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
addpd	ADDPD	add packed double-precision floating-point values	
addsd	ADDSD	add scalar double-precision floating-point values	
divpd	DIVPD	divide packed double-precision floating-point values	
divsd	DIVSD	divide scalar double-precision floating-point values	
maxpd	MAXPD	return maximum packed double-precision floating-point values	
maxsd	MAXSD	return maximum scalar double-precision floating-point value	
minpd	MINPD	return minimum packed double-precision floating-point values	
minsd	MINSD	return minimum scalar double-precision floating-point value	
mulpd	MULPD	multiply packed double-precision floating-point values	
mulsd	MULSD	multiply scalar double-precision floating-point values	
sqrtpd	SQRTPD	compute packed square roots of packed double-precision floating-point values	

TABLE 3–37 SSE2 Packed Arithmetic Instructions *(Continued)*

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>sqrtsd</code>	<code>SQRTSD</code>	compute scalar square root of scalar double-precision floating-point value	
<code>subpd</code>	<code>SUBPD</code>	subtract packed double-precision floating-point values	
<code>subsd</code>	<code>SUBSD</code>	subtract scalar double-precision floating-point values	

SSE2 Logical Instructions

The SSE2 logical instructions operate on packed double-precision floating-point values.

TABLE 3–38 SSE2 Logical Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>andnpd</code>	<code>ANDNPD</code>	perform bitwise logical AND NOT of packed double-precision floating-point values	
<code>andpd</code>	<code>ANDPD</code>	perform bitwise logical AND of packed double-precision floating-point values	
<code>orpd</code>	<code>ORPD</code>	perform bitwise logical OR of packed double-precision floating-point values	
<code>xorpd</code>	<code>XORPD</code>	perform bitwise logical XOR of packed double-precision floating-point values	

SSE2 Compare Instructions

The SSE2 compare instructions compare packed and scalar double-precision floating-point values and return the results of the comparison to either the destination operand or to the EFLAGS register.

TABLE 3–39 SSE2 Compare Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cmpdpd	CMPPD	compare packed double-precision floating-point values	
cmpsd	CMPSD	compare scalar double-precision floating-point values	
comisd	COMISD	perform ordered comparison of scalar double-precision floating-point values and set flags in EFLAGS register	
ucomisd	UCOMISD	perform unordered comparison of scalar double-precision floating-point values and set flags in EFLAGS register	

SSE2 Shuffle and Unpack Instructions

The SSE2 shuffle and unpack instructions operate on packed double-precision floating-point operands.

TABLE 3–40 SSE2 Shuffle and Unpack Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
shufpd	SHUFPPD	shuffle values in packed double-precision floating-point operands	
unpckhpd	UNPCKHPD	unpack and interleave the high values from two packed double-precision floating-point operands	

TABLE 3-40 SSE2 Shuffle and Unpack Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
unpcklpd	UNPCKLPD	unpack and interleave the low values from two packed double-precision floating-point operands	

SSE2 Conversion Instructions

The SSE2 conversion instructions convert packed and individual doubleword integers into packed and scalar double-precision floating-point values (and vice versa). These instructions also convert between packed and scalar single-precision and double-precision floating-point values.

TABLE 3-41 SSE2 Conversion Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cvt dq2pd	CVTDQ2PD	convert packed doubleword integers to packed double-precision floating-point values	
cvt pd2dq	CVTPD2DQ	convert packed double-precision floating-point values to packed doubleword integers	
cvt pd2pi	CVTPD2PI	convert packed double-precision floating-point values to packed doubleword integers	
cvt pd2ps	CVTPD2PS	convert packed double-precision floating-point values to packed single-precision floating-point values	

TABLE 3-41 SSE2 Conversion Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cvtpi2pd	CVTPI2PD	convert packed doubleword integers to packed double-precision floating-point values	
cvtps2pd	CVTPS2PD	convert packed single-precision floating-point values to packed double-precision floating-point values	
cvtsd2si	CVTSD2SI	convert scalar double-precision floating-point values to a doubleword integer	
cvtsd2ss	CVTSD2SS	convert scalar double-precision floating-point values to scalar single-precision floating-point values	
cvtsi2sd	CVTSI2SD	convert doubleword integer to scalar double-precision floating-point value	
cvtss2sd	CVTSS2SD	convert scalar single-precision floating-point values to scalar double-precision floating-point values	
cvttpd2dq	CVTPD2DQ	convert with truncation packed double-precision floating-point values to packed doubleword integers	

TABLE 3-41 SSE2 Conversion Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cvttpd2pi	CVTTPD2PI	convert with truncation packed double-precision floating-point values to packed doubleword integers	
cvttsd2si	CVTTSD2SI	convert with truncation scalar double-precision floating-point values to scalar doubleword integers	

SSE2 Packed Single-Precision Floating-Point Instructions

The SSE2 packed single-precision floating-point instructions operate on single-precision floating-point and integer operands.

TABLE 3-42 SSE2 Packed Single-Precision Floating-Point Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
cvtdq2ps	CVTDQ2PS	convert packed doubleword integers to packed single-precision floating-point values	
cvtps2dq	CVTPS2DQ	convert packed single-precision floating-point values to packed doubleword integers	
cvttps2dq	CVTTPS2DQ	convert with truncation packed single-precision floating-point values to packed doubleword integers	

SSE2 128-Bit SIMD Integer Instructions

The SSE2 SIMD integer instructions operate on packed words, doublewords, and quadwords contained in XMM and MMX registers.

TABLE 3-43 SSE2 128-Bit SIMD Integer Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
movdq2q	MOVDQ2Q	move quadword integer from XMM to MMX registers	
movdqa	MOVDQA	move aligned double quadword	
movdqu	MOVDQU	move unaligned double quadword	
movq2dq	MOVQ2DQ	move quadword integer from MMX to XMM registers	
paddq	PADDQ	add packed quadword integers	
pmuludq	PMULUDQ	multiply packed unsigned doubleword integers	
pshufd	PSHUFD	shuffle packed doublewords	
pshufhw	PSHUFHW	shuffle packed high words	
pshuflw	PSHUFLW	shuffle packed low words	
pslldq	PSLLDQ	shift double quadword left logical	
psrldq	PSRLDQ	shift double quadword right logical	
psubq	PSUBQ	subtract packed quadword integers	
punpckhqdq	PUNPCKHQDQ	unpack high quadwords	
punpcklqdq	PUNPCKLQDQ	unpack low quadwords	

SSE2 Miscellaneous Instructions

The SSE2 instructions described below provide additional functionality for caching non-temporal data when storing data from XMM registers to memory, and provide additional control of instruction ordering on store operations.

TABLE 3-44 SSE2 Miscellaneous Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>clflush</code>	<code>CLFLUSH</code>	flushes and invalidates a memory operand and its associated cache line from all levels of the processor's cache hierarchy	
<code>lfence</code>	<code>LFENCE</code>	serializes load operations	
<code>maskmovdqu</code>	<code>MASKMOVDQU</code>	non-temporal store of selected bytes from an XMM register into memory	
<code>mfence</code>	<code>MFENCE</code>	serializes load and store operations	
<code>movntdq</code>	<code>MOVNTDQ</code>	non-temporal store of double quadword from an XMM register into memory	
<code>movnti</code>	<code>MOVNTI</code>	non-temporal store of a doubleword from a general-purpose register into memory	<code>movntiq</code> valid only under <code>-xarch=amd64</code>
<code>movntpd</code>	<code>MOVNTPD</code>	non-temporal store of two packed double-precision floating-point values from an XMM register into memory	
<code>pause</code>	<code>PAUSE</code>	improves the performance of spin-wait loops	

Operating System Support Instructions

The operating system support instructions provide functionality for process management, performance monitoring, debugging, and other systems tasks.

TABLE 3-45 Operating System Support Instructions

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
arpl	ARPL	adjust requested privilege level	
clts	CLTS	clear the task-switched flag	
hlt	HLT	halt processor	
invd	INVD	invalidate cache, no writeback	
invlpg	INVLPG	invalidate TLB entry	
lar	LAR	load access rights	larq valid only under -xarch=amd64
lgdt	LGDT	load global descriptor table (GDT) register	
lidt	LIDT	load interrupt descriptor table (IDT) register	
lldt	LLDT	load local descriptor table (LDT) register	
lmsw	LMSW	load machine status word	
lock	LOCK	lock bus	
lsl	LSL	load segment limit	lslq valid only under -xarch=amd64
ltr	LTR	load task register	
rdmsr	RDMSR	read model-specific register	
rdpmc	RDPMC	read performance monitoring counters	

TABLE 3-45 Operating System Support Instructions (Continued)

Solaris Mnemonic	Intel/AMD Mnemonic	Description	Notes
<code>rdtsc</code>	RDTSC	read time stamp counter	
<code>rsm</code>	RSM	return from system management mode (SMM)	
<code>sgdt</code>	SGDT	store global descriptor table (GDT) register	
<code>sidt</code>	SIDT	store interrupt descriptor table (IDT) register	
<code>sldt</code>	SLDT	store local descriptor table (LDT) register	<code>sldtq</code> valid only under -xarch=amd64
<code>smsw</code>	SMSW	store machine status word	<code>smswq</code> valid only under -xarch=amd64
<code>str</code>	STR	store task register	<code>strq</code> valid only under -xarch=amd64
<code>sysenter</code>	SYSENTER	fast system call, transfers to a flat protected model kernel at CPL=0	
<code>sysexit</code>	SYSEXIT	fast system call, transfers to a flat protected mode kernel at CPL=3	
<code>verr</code>	VERR	verify segment for reading	
<code>verw</code>	VERW	verify segment for writing	
<code>wbinvd</code>	WBINVD	invalidate cache, with writeback	
<code>wrmsr</code>	WRMSR	write model-specific register	

64–Bit AMD Opteron Considerations

To assemble code for the AMD Opteron CPU, invoke the assembler with the `-xarch=amd64` command line option. See the `as(1)` man page for additional information.

The following Solaris mnemonics are only valid when the `-xarch=amd64` command line option is specified:

<code>adcq</code>	<code>cmovnoq</code>	<code>mulq</code>
<code>addq</code>	<code>cmovnpq</code>	<code>negq</code>
<code>andq</code>	<code>cmovnsq</code>	<code>notq</code>
<code>bsfq</code>	<code>cmovnzq</code>	<code>orq</code>
<code>bsrq</code>	<code>cmovoq</code>	<code>popfq</code>
<code>bswapq</code>	<code>cmovpeq</code>	<code>popq</code>
<code>btcq</code>	<code>cmovpoq</code>	<code>pushfq</code>
<code>btq</code>	<code>cmovpq</code>	<code>pushq</code>
<code>btrq</code>	<code>cmovsq</code>	<code>rclq</code>
<code>btsq</code>	<code>cmovzq</code>	<code>rcrq</code>
<code>cltq</code>	<code>cmpq</code>	<code>rolq</code>
<code>cmovaeq</code>	<code>cmpsq</code>	<code>rorq</code>
<code>cmovaq</code>	<code>cmpxchgq</code>	<code>salq</code>
<code>cmovbeq</code>	<code>cqtd</code>	<code>sarq</code>
<code>cmovbq</code>	<code>cqto</code>	<code>sbbq</code>
<code>cmovcq</code>	<code>decq</code>	<code>scasq</code>
<code>cmoveq</code>	<code>divq</code>	<code>shldq</code>
<code>cmovgeq</code>	<code>idivq</code>	<code>shlq</code>
<code>cmovgq</code>	<code>imulq</code>	<code>shrdq</code>
<code>cmovleq</code>	<code>incq</code>	<code>shrq</code>
<code>cmovlq</code>	<code>larq</code>	<code>sldtq</code>
<code>cmovnaeq</code>	<code>leaq</code>	<code>smswq</code>
<code>cmovnaq</code>	<code>lodsq</code>	<code>stosq</code>
<code>cmovnbeq</code>	<code>lslq</code>	<code>strq</code>
<code>cmovnbq</code>	<code>movabs</code>	<code>subq</code>
<code>cmovncq</code>	<code>movdq</code>	<code>testq</code>
<code>cmovneq</code>	<code>movntiq</code>	<code>xaddq</code>
<code>cmovngeq</code>	<code>movq</code>	<code>xchgq</code>
<code>cmovngq</code>	<code>movsq</code>	<code>xchgqA</code>
<code>cmovnlq</code>	<code>movswq</code>	<code>xorq</code>
<code>cmovnlq</code>	<code>movzsq</code>	

The following Solaris mnemonics are *not* valid when the `-xarch=amd64` command line option is specified:

<code>aaa</code>	<code>aas</code>	<code>das</code>
<code>aad</code>	<code>boundw</code>	<code>into</code>
<code>aam</code>	<code>daa</code>	<code>jecxz</code>

ldsw
lesw

popa
popaw

pusha
pushaw

Index

A

aaa, 32
aad, 32
aam, 32
aas, 32
adc, 30
add, 30
addpd, 65
addps, 56
addressing, 19
addsd, 65
addss, 56
.align, 20
and, 32
andnpd, 66
andnps, 59
andpd, 66
andps, 59
arpl, 73
as, 11
 command line, 12
 ELF object file, 11
 macro processing, 11
 syntax, UNIX versus Intel, 12
.ascii, 20
assembler, *See* as

B

.bcd, 20
binary arithmetic instructions, 30
bit instructions, 33

bound, 36
bsf, 34
bsr, 34
.bss, 20
bswap, 26
bt, 34
btc, 34
btr, 34
bts, 34
.2byte, 20
.4byte, 20
.8byte, 20
.byte, 20
byte instructions, 33

C

call, 36
cbtw, 26
clc, 40
cld, 40
clflush, 72
cli, 40
cltd, 26
cltq, 26
clts, 73
cmc, 40
cmov.a, 26
cmova, 26
cmov.ae, 26
cmovae, 26
cmov.b, 26

cmovb, 26
 cmov.be, 27
 cmovbe, 27
 cmov.c, 27
 cmovc, 27
 cmov.e, 27
 cmove, 27
 cmov.g, 27
 cmovg, 27
 cmov.ge, 27
 cmovge, 27
 cmov.l, 27
 cmovl, 27
 cmov.le, 27
 cmovle, 27
 cmov.na, 27
 cmovna, 27
 cmov.nae, 27
 cmovnae, 27
 cmov.nb, 27
 cmovnb, 27
 cmov.nbe, 27
 cmovnbe, 27
 cmov.nc, 27
 cmovnc, 27
 cmov.ne, 27
 cmovne, 27
 cmov.ng, 28
 cmovng, 28
 cmov.nge, 28
 cmovnge, 28
 cmov.nl, 28
 cmovnl, 28
 cmov.nle, 28
 cmovnle, 28
 cmov.no, 28
 cmovno, 28
 cmov.np, 28
 cmovnp, 28
 cmov.ns, 28
 cmovns, 28
 cmov.nz, 28
 cmovnz, 28
 cmov.o, 28
 cmovo, 28
 cmov.p, 28
 cmovp, 28
 cmovpe, 28
 cmovpo, 28
 cmovs, 28
 cmovz, 29
 cmp, 31
 cmppd, 67
 cmpps, 58
 cmps, 38
 cmpsb, 38
 cmpsd, 67
 cmpsl, 38
 cmpss, 58
 cmpsw, 38
 cmpxchg, 29
 cmpxchg8b, 29
 comisd, 67
 comiss, 58
 .comm, 20
 comment, 13
 control transfer instructions, 35
 cpp, 11
 cpuid, 41
 cqtd, 29
 cqto, 29
 cvtdq2pd, 68
 cvtdq2ps, 70
 cvtpd2dq, 68
 cvtpd2pi, 68
 cvtpd2ps, 68
 cvtpi2pd, 69
 cvtpi2ps, 60
 cvtps2dq, 70
 cvtps2pd, 69
 cvtps2pi, 60
 cvtsd2si, 69
 cvtsd2ss, 69
 cvtsi2sd, 69
 cvtsi2ss, 60
 cvtss2sd, 69
 cvtss2si, 60
 cvttpd2dq, 69
 cvttpd2pi, 70
 cvttps2dq, 70
 cvttps2pi, 61
 cvttsd2si, 70
 cvttss2si, 61
 cwt, 29
 cwtl, 29

D

daa, 32
das, 32
.data, 20
data transfer instructions, 26
dec, 31
decimal arithmetic instructions, 31
directives, 20
div, 31
divpd, 65
divps, 56
divsd, 65
divss, 57
.double, 20

E

ELF object file, 11
emms, 54
enter, 36
.even, 21
.ext, 21

F

f2xm1, 46
fabs, 44
fadd, 44
faddp, 44
fbs, *See as*
fbld, 42
fbstp, 42
fchs, 44
fclex, 47
fcmovb, 42
fcmovbe, 42
fcmovl, 43
fcmovnb, 43
fcmovnbe, 43
fcmovne, 43
fcmovnu, 43
fcmovu, 43
fcom, 45
fcomi, 45
fcomip, 45
fcomp, 45

fcompp, 45
fcos, 46
fdecstp, 47
fdiv, 44
fdivp, 44
fdivr, 44
fdivrp, 44
ffree, 47
fiadd, 44
ficom, 45
ficomp, 45
fidiv, 44
fidivr, 44
fild, 43
.file, 21
fimul, 44
fincstp, 47
finit, 48
fist, 43
fistp, 43
fisub, 44
fisubr, 44
flag control instructions, 40
fld, 43
fldl, 47
fldcw, 48
fldenv, 48
fldl2e, 47
fldl2t, 47
fldlg2, 47
fldln2, 47
fldpi, 47
fldz, 47
.float, 21
floating-point instructions
 basic arithmetic, 43
 comparison, 45
 control, 47
 data transfer, 42
 load constants, 47
 logarithmic
 See transcendental
 transcendental, 46
 trigonometric
 See transcendental
fmul, 44
fmulp, 44
fnclex, 48

fninit, 48
 fnop, 48
 fnsave, 48
 fnstcw, 48
 fnstenv, 48
 fnstsw, 48
 fpatan, 46
 fprem, 44
 fprem1, 44
 fptan, 46
 frndint, 44
 frstor, 48
 fsave, 48
 fscale, 44
 fsin, 46
 fsincos, 46
 fsqrt, 44
 fst, 43
 fstcw, 49
 fstenv, 49
 fstp, 43
 fstsw, 49
 fsub, 44
 fsubp, 45
 fsubr, 45
 fsubrp, 45
 ftst, 45
 fucom, 46
 fucomi, 46
 fucomip, 46
 fucomp, 46
 fucompp, 46
 fwait, 49
 fxam, 46
 fxch, 43
 fxrstor, 49
 fxsave, 49
 fextract, 45
 fyl2x, 46
 fyl2xp1, 46

G

gas, 12
 .globl, 21
 .group, 21

H

.hidden, 21
 hlt, 73

I

I/O (input/output) instructions, 39
 .ident, 21
 identifier, 15
 idiv, 31
 imul, 31
 in, 40
 inc, 31
 ins, 40
 insb, 40
 insl, 40
 instruction, 17
 format, 17
 suffixes, 18
 instructions
 binary arithmetic, 30
 bit, 33
 byte, 33
 control transfer, 35
 data transfer, 26
 decimal arithmetic, 31
 flag control, 40
 floating-point, 42-49
 I/O (input/output), 39
 logical, 32
 miscellaneous, 41
 MMX, 49-54
 operating system support, 73-74
 Opteron, 75
 rotate, 32
 segment register, 41
 shift, 32
 SIMD state management, 49
 SSE, 54-63
 SSE2, 63-72
 string, 38
 insw, 40
 int, 36
 into, 36
 invd, 73
 invlpg, 73
 iret, 36

J

ja, 36
jae, 36
jb, 36
jbe, 36
jc, 36
jcxz, 36
je, 36
jecxz, 36
jg, 36
jge, 36
jl, 36
jle, 36
jmp, 36
jnae, 36
jnb, 36
jnbe, 36
jnc, 36
jne, 37
jng, 37
jnge, 37
jnl, 37
jnle, 37
jno, 37
jnp, 37
jns, 37
jnz, 37
jo, 37
jp, 37
jpe, 37
jpo, 37
js, 37
jz, 37

K

keyword, 15

L

label, 14
 numeric, 14
 symbolic, 14
lahf, 40
lar, 73
lcall, 37

.lcomm, 21
ldmxcsr, 61
lds, 41
lea, 41
leave, 37
les, 41
lfence, 72
lfs, 41
lgdt, 73
lgs, 41
lidt, 73
lldt, 73
lmsw, 73
.local, 21
lock, 73
lods, 38
lodsb, 38
lodsl, 38
lodsw, 38
logical instructions, 32
.long, 22
loop, 37
loope, 37
loopne, 37
loopnz, 37
loopz, 37
lret, 38
lsl, 73
lss, 41
ltr, 73

M

m4, 11
maskmovdqu, 72
maskmovq, 62
maxpd, 65
maxps, 57
maxsd, 65
maxss, 57
mfence, 72
minpd, 65
minps, 57
minsd, 65
minss, 57
miscellaneous instructions, 41

- MMX instructions
 - comparison, 52
 - conversion, 50
 - data transfer, 50
 - logical, 53
 - packed arithmetic, 51
 - rotate, 53
 - shift, 53
 - state management, 54
- mov, 29
- movabs, 29
- movabsA, 29
- movapd, 64
- movaps, 55
- movd, 50
- movdq2q, 71
- movdqa, 71
- movdqu, 71
- movhlps, 55
- movhpd, 64
- movhps, 55
- movlhps, 55
- movlpd, 64
- movlps, 56
- movmskpd, 64
- movmskps, 56
- movntdq, 72
- movnti, 72
- movntpd, 72
- movntps, 62
- movntq, 63
- movq, 50
- movq2dq, 71
- movs, 38
- movsb, 29, 39
- movsd, 64
- movsl, 39
- movss, 56
- movsw, 29, 39
- movupd, 64
- movups, 56
- movzb, 29
- movzw, 29
- mul, 31
- mulpd, 65
- mulps, 57
- mulsd, 65
- mulss, 57

N

- neg, 31
- nop, 41
- not, 32
- numbers, 15
 - floating point, 16
 - integers, 15
 - binary, 15
 - decimal, 15
 - hexadecimal, 15
 - octal, 15

O

- operands, 18
 - immediate, 18
 - indirect, 18
 - memory
 - addressing, 19
 - ordering (source, destination), 18
 - register, 18
- operating system support instructions, 73
- Opteron instructions, 75
- or, 32
- orpd, 66
- orps, 59
- out, 40
- outs, 40
- outsb, 40
- outsl, 40
- outsw, 40

P

- packssdw, 50
- packsswb, 50
- packuswb, 50
- paddb, 51
- paddd, 51
- paddq, 71
- paddsb, 51
- paddsw, 51
- paddusb, 51
- paddusw, 51
- paddw, 51
- pand, 53

pandn, 53
pause, 72
pavgb, 61
pavgw, 61
pcmpeqb, 52
pcmpeqd, 52
pcmpeqw, 52
pcmpgtb, 52
pcmpgtd, 53
pcmpgtw, 53
pextrw, 62
pinsrw, 62
pmaddwd, 51
pmaxsw, 62
pmaxub, 62
pminsw, 62
pminub, 62
pmovmskb, 62
pmulhw, 62
pmulhw, 51
pmullw, 51
pmuludq, 71
pop, 30
popa, 30
popal, 30
popaw, 30
popf, 41
popfw, 40
.popsection, 22
por, 53
prefetchnta, 63
prefetcht0, 63
prefetcht1, 63
prefetcht2, 63
.previous, 22
psadbw, 62
pshufd, 71
pshufhw, 71
pshuflw, 71
pshufw, 62
pslld, 53
pslldq, 71
psllq, 53
psllw, 53
psrad, 53
psraw, 53
psrld, 53
psrldq, 71

psrlq, 53
psrlw, 53
psubb, 52
psubd, 52
psubq, 71
psubsb, 52
psubsw, 52
psubusb, 52
psubusw, 52
psubw, 52
punpckhbw, 50
punpckhdq, 50
punpckhqdq, 71
punpckhwd, 50
punpcklbw, 50
punpckldq, 50
punpcklqdq, 71
punpcklwd, 51
push, 30
pusha, 30
pushal, 30
pushaw, 30
pushf, 41
pushfw, 41
.pushsection, 22
pxor, 53

Q

.quad, 22

R

rcl, 33
rcpps, 57
rcpss, 57
rcr, 33
rdmsr, 73
rdpmc, 73
rdtsc, 74
.rel, 22
rep, 39
repnz, 39
repz, 39
ret, 38
rol, 33

ror, 33
rotate instructions, 32
rsm, 74
rsqrtps, 57
rsqrtss, 57

S

sahf, 41
sal, 33
sar, 33
sbb, 31
scas, 39
scasb, 39
scasl, 39
scasw, 39
.section, 22
segment register instructions, 41
.set, 22
seta, 34
setae, 34
setb, 34
setbe, 34
setc, 34
sete, 34
setg, 34
setge, 34
setl, 34
setle, 34
setna, 34
setnae, 34
setnb, 34
setnbe, 35
setnc, 35
setne, 35
setng, 35
setnge, 35
setnl, 35
setnle, 35
setno, 35
setnp, 35
setns, 35
setnz, 35
seto, 35
setp, 35
setpe, 35
setpo, 35

sets, 35
setz, 35
sfence, 63
sgdt, 74
shift instructions, 32
shl, 33
shld, 33
shr, 33
shrd, 33
shufpd, 67
shufps, 59
sidt, 74
SIMD state management instructions, 49
.skip, 22
sldt, 74
.sleb128, 22
smovl, 39
smsw, 74
sqrtpd, 65
sqrtps, 58
sqrtsd, 66
sqrtss, 58
SSE instructions
 compare, 58
 conversion, 60
 data transfer, 55
 integer (64-bit SIMD), 61
 logical, 59
 miscellaneous, 62
 MXCSR state management, 61
 packed arithmetic, 56
 shuffle, 59
 unpack, 59
SSE2 instructions
 compare, 66
 conversion, 68
 data movement, 64
 logical, 66
 miscellaneous, 72
 packed arithmetic, 64
 packed single-precision floating-point, 70
 shuffle, 67
 SIMD integer instructions (128-bit), 70
 unpack, 67
statement, 13
 empty, 13
stc, 41
std, 41

sti, 41
stmxcsr, 61
stos, 39
stosb, 39
stosl, 39
stosw, 39
str, 74
.string, 23
string, 16
string instructions, 38
sub, 31
subpd, 66
subps, 58
subsd, 66
subss, 58
.symbolic, 23
sysenter, 74
sysexit, 74

T

.tbss, 23
.tcomm, 23
.tdata, 23
test, 35
.text, 23

U

ucomisd, 67
ucomiss, 58
ud2, 41
.uleb128, 23
unpckhpd, 67
unpckhps, 60
unpcklpd, 68
unpcklps, 60

V

.value, 23
verr, 74
verw, 74

W

wait, 49
wbinvd, 74
.weak, 23
whitespace, 13
wrmsr, 74

X

xadd, 30
xchg, 30
xchgA, 30
xlat, 41
xlatb, 41
xor, 32
xorpd, 66
xorps, 59

Z

.zero, 23

