
Variable-Length Encoding (VLE) Programming Environments Manual:

A Supplement to the EREF

VLEPEM
Rev. 0
07/2007



How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc., 2007. All rights reserved.



Contents

Paragraph Number	Title	Page Number
About This Book		
Chapter 1		
Introduction		
1.1	Overview	1-1
1.2	Documentation Conventions	1-1
1.2.1	Description of Instruction Operation	1-2
1.3	Instruction Mnemonics and Operands	1-2
Chapter 2		
Instruction Model		
2.1	VLE Storage Addressing	2-1
2.1.1	Data Storage Addressing Modes	2-1
2.1.2	Instruction Storage Addressing Modes	2-2
2.1.2.1	Misaligned, Mismatched, and Byte-Ordering Instruction Storage Exceptions	2-2
2.1.2.2	VLE Exception Syndrome Bits	2-3
2.2	VLE Compatibility with the Standard Architecture	2-3
2.2.1	Overview	2-3
2.2.2	VLE Processor and Storage Control Extensions	2-3
2.2.2.1	Instruction Extensions	2-4
2.2.2.2	MMU Extensions	2-4
2.2.3	VLE Limitations	2-4
2.3	Branch Operation Instructions	2-4
2.3.1	Branch Processor Registers	2-4
2.3.1.1	Condition Register (CR)	2-5
2.3.1.1.1	Condition Register Setting for Compare Instructions	2-6
2.3.1.1.2	Condition Register Setting for the Bit Test Instruction	2-6
2.3.1.2	Link Register (LR)	2-6
2.3.1.3	Count Register (CTR)	2-6
2.3.2	Branch Instructions	2-6
2.3.3	System Linkage Instructions	2-7
2.4	Integer Instructions	2-8
2.4.1	Integer Load Instructions	2-8
2.4.2	Integer Load and Store with Byte Reversal Instructions	2-8
2.4.3	Integer Load and Store Multiple Instructions	2-8
2.4.4	Integer Arithmetic Instructions	2-8
2.4.5	Integer Trap Instructions	2-9
2.4.6	Integer Select Instruction	2-9
2.4.7	Integer Logical, Bit, and Move Instructions	2-9

Contents

Paragraph Number	Title	Page Number
2.5	Storage Control Instructions	2-10
2.5.1	Storage Synchronization Instructions	2-10
2.5.2	Cache Management Instructions.....	2-10
2.5.3	Cache Locking Instructions	2-11
2.5.4	TLB Management Instructions	2-11
2.5.5	Instruction Alignment and Byte Ordering	2-11
2.6	Additional Categories Available in VLE	2-11

Chapter 3 VLE Instruction Set

3.1	Supported Power ISA Instructions	3-1
3.2	Immediate Field and Displacement Field Encodings	3-4

Appendix A VLE Instruction Formats

A.1	Overview	A-1
A.2	VLE Instruction Formats	A-1
A.2.1	Instruction Fields	A-1
A.2.2	BD8 Form (16-Bit Branch Instructions).....	A-3
A.3	C Form (16-Bit Control Instructions).....	A-3
A.4	IM5 Form (16-Bit register + immediate Instructions)	A-4
A.5	OIM5 Form (16-Bit Register + Offset Immediate Instructions).....	A-4
A.6	IM7 Form (16-Bit Load immediate Instructions)	A-4
A.7	R Form (16-Bit Monadic Instructions)	A-4
A.8	RR Form (16-Bit Dyadic Instructions)	A-4
A.9	SD4 Form (16-Bit Load/Store Instructions)	A-5
A.10	BD15 Form	A-5
A.11	BD24 Form	A-5
A.12	D8 Form	A-5
A.13	I16A Form.....	A-5
A.14	I16L Form	A-6
A.15	M Form	A-6
A.16	SCI8 Form.....	A-6
A.17	LI20 Form	A-6

Appendix B VLE Instruction Set Tables

Contents

Paragraph Number	Title	Page Number
B.1	VLE Instruction Set Sorted by Mnemonic	B-1
B.2	VLE Instruction Set Sorted by Opcode	B-27



Contents

**Paragraph
Number**

Title

**Page
Number**

Figures

Figure Number	Title	Page Number
A-1	BD8 Instruction Format	A-3
A-2	C Instruction Format	A-3
A-3	IM5 Instruction Format	A-4
A-4	OIM5 Instruction Format	A-4
A-5	IM7 Instruction Format	A-4
A-6	R Instruction Format	A-4
A-7	RR Instruction Format	A-4
A-8	SD4 Instruction Format.....	A-5
A-9	BD15 Instruction Format	A-5
A-10	BD24 Instruction Format	A-5
A-11	D8 Instruction Format	A-5
A-12	I16A Instruction Format.....	A-5
A-13	I16L Instruction Format	A-6
A-14	M Instruction Format	A-6
A-15	SC18 Instruction Format.....	A-6
A-16	LI20 Instruction Format	A-6



Figures

**Figure
Number**

Title

**Page
Number**

Tables

Table Number	Title	Page Number
2-1	Data Storage Addressing Modes	2-1
2-2	Instruction Storage Addressing Modes	2-2
2-3	CR0 Field Descriptions	2-5
2-4	Condition Register Settings for Compare Instructions	2-6
2-5	BO32 Field Encodings	2-7
2-6	BO16 Field Encodings	2-7
3-1	Non-VLE Instructions Listed by Mnemonic	3-1
3-2	Immediate Field and Displacement Field Encodings	3-4
A-1	Instruction Fields.....	A-1
B-1	Mode Dependency and Privilege Abbreviations	B-1
B-2	VLE Instruction Set Sorted by Mnemonic.....	B-1
B-3	VLE Instruction Set Sorted by Opcode.....	B-27



Tables

**Table
Number**

Title

**Page
Number**

About This Book

The primary objective of this manual is to help programmers provide software that is compatible with processors that implement the VLE category.

To locate any published errata or updates for this document, refer to the web at <http://www.freescale.com>.

This book is used as a reference guide for assembler programmers. It uses a standardized format instruction to describe each instruction, showing syntax, instruction format, register translation language (RTL) code that describes how the instruction works, and a listing of which, if any, registers are affected. At the bottom of each instruction entry is a figure that shows the operations on elements within source operands and where the results of those operations are placed in the destination operand.

The *VLE Programming Interface Manual* (VLE PIM) is a reference guide for high-level programmers. The VLE PIM describes how programmers can access VLE functionality from programming languages such as C and C++. It defines a programming model for use with the VLE instruction set. Processors that implement the Power ISA use the VLE instruction set as an extension to the base and embedded categories of the Power ISA.

Because it is important to distinguish between the categories of the Power ISA to ensure compatibility across multiple platforms, those distinctions are shown clearly throughout this book. This document stays consistent with the Power ISA in referring to three levels, or programming environments, which are as follows:

- User instruction set architecture (UISA)—The UISA defines the level of the architecture to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, memory conventions, and the memory and programming models seen by application programmers.
- Virtual environment architecture (VEA)—The VEA, which is the smallest component of the Power architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple processors or other devices can access external memory and defines aspects of the cache model and cache control instructions from a user-level perspective. VEA resources are particularly useful for optimizing memory accesses and for managing resources in an environment in which other processors and other devices can access external memory.

Implementations that conform to the VEA also conform to the UISA but may not necessarily adhere to the OEA.

- Operating environment architecture (OEA)—The OEA defines supervisor-level resources typically required by an operating system. It defines the memory management model, supervisor-level registers, and the exception model.

Implementations that conform to the OEA also conform to the UISA and VEA.

Most of the discussions on the VLE are at the UISA level. For ease in reference, this book and the processor reference manuals have arranged the architecture information into topics that build on one another, beginning with a description and complete summary of registers and instructions (for all three environments) and progressing to more specialized topics such as the cache, exception, and memory management models. As such, chapters may include information from multiple levels of the architecture, but when discussing OEA and VEA, the level is noted in the text.

It is beyond the scope of this manual to describe individual devices that implement VLE. It must be kept in mind that each processor that implements the Power ISA is unique in its implementation.

The information in this book is subject to change without notice, as described in the disclaimers on the title page of this book. As with any technical documentation, it is the readers' responsibility to be sure they are using the most recent version of the documentation. For more information, contact your sales representative or visit our web site at <http://www.freescale.com>.

Audience

This manual is intended for system software and hardware developers, and for application programmers who want to develop products using the VLE. It is assumed that the reader understands operating systems, microprocessor system design, and the basic principles of RISC processing and details of the Power ISA.

This book describes how VLE interacts with the other components of the Power architecture.

Organization

Following is a summary and a brief description of the major sections of this manual:

- [Chapter 1, “Introduction,”](#) is useful for those who want a general understanding of the features and functions of the VLE. This chapter provides an overview of how the VLE defines the register set, operand conventions, addressing modes, instruction set, and interrupt model.
- [Chapter 2, “Instruction Model,”](#) describes the VLE instruction set, including operand conventions, addressing modes, and instruction syntax. It also provides a brief description of the VLE instructions organized by function.
- [Chapter 3, “VLE Instruction Set,”](#) functions as a handbook for the VLE instruction set. Instructions are sorted by mnemonic. Each instruction description includes the instruction formats and figures where it helps in understanding what the instruction does.
- [Appendix A, “VLE Instruction Formats,”](#) lists all of the VLE formats, grouped according to mnemonic, opcode, and form, in both decimal and binary order.
- [Appendix B, “VLE Instruction Set Tables,”](#) lists all VLE instructions, grouped according to mnemonic and opcode.
- This manual also includes an index.

Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the VLE and the Power ISA.

General Information

The following documentation provides useful information about the PowerPC architecture and computer architecture in general:

- *Computer Architecture: A Quantitative Approach*, Third Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Third Edition, David A. Patterson and John L. Hennessy.

Related Documentation

Freescale documentation is available from the sources listed on the back of the title page; the document order numbers, when applicable, are included in parentheses for ease in ordering:

- *EREF: A Programmer's Reference Manual for Freescale Embedded Processors* (EREFM). Describes the programming, memory management, cache, and interrupt models defined by the Power ISA for embedded environment processors.
 - *Power ISA™*, The latest version of the Power instruction set architecture can be downloaded from the website www.power.org.
 - *VLE Programming Interface Manual* (VLEPIM). Provides the VLE-specific extensions to the e500 application binary interface.
 - *e500 Application Binary Interface User's Guide* (E500ABIUG). Establishes a standard binary interface for application programs on systems that implement the interfaces defined in the System V Interface Definition, Issue 3. This includes systems that have implemented UNIX System V Release 4.
 - Reference manuals. The following reference manuals provide details information about processor cores and integrated devices:
 - Core reference manuals—These books describe the features and behavior of individual microprocessor cores and provide specific information about how functionality described in the EREF is implemented by a particular core. They also describe implementation-specific features and microarchitectural details, such as instruction timing and cache hardware details, that lie outside the architecture specification.
 - Integrated device reference manuals—These manuals describe the features and behavior of integrated devices that implement a Power ISA processor core. It is important to understand that some features defined for a core may not be supported on all devices that implement that core.
- Also, some features are defined in a general way at the core level and have meaning only in the context of how the core is implemented. For example, any implementation-specific behavior of register fields can be described only in the reference manual for the integrated device.
- Each of these documents include the following two chapters that are pertinent to the core:
- A core overview. This chapter provides a general overview of how the core works and indicates which of a core's features are implemented on the integrated device.
 - A register summary chapter. This chapter gives the most specific information about how register fields can be interpreted in the context of the implementation.

These reference manuals also describe how the core interacts with other blocks on the integrated device, especially regarding topics such as reset, interrupt controllers, memory and cache management, debug, and global utilities.

- Addenda/errata to reference manuals—Errata documents are provided to address errors in published documents.

Because some processors have follow-on parts, often an addendum is provided that describes the additional features and functionality changes. These addenda, which may also contain errata, are intended for use with the corresponding reference manuals.

Always check the Freescale website for updates to reference manuals.

- Hardware specifications—Hardware specifications provide specific data regarding bus timing; signal behavior; AC, DC, and thermal characteristics; and other design considerations.
- Product brief—Each integrated device has a product brief that provides an overview of its features. This document is roughly the equivalent to the overview (Chapter 1) of the device’s reference manual.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

Additional literature is published as new processors become available. For current documentation, refer to <http://www.freescale.com>.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
mnemonics	Instruction mnemonics are shown in lowercase bold
<i>italics</i>	Italics indicate variable command parameters, for example, bcctrx Book titles in text are set in italics
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, rB	Instruction syntax used to identify a source general-purpose register (GPR)
rD	Instruction syntax used to identify a destination GPR
frA, frB, frC	Instruction syntax used to identify a source floating-point register (FPR)
frD	Instruction syntax used to identify a destination FPR
REG[FIELD]	Abbreviations for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.
x	In some contexts, such as signal encodings, an unitalicized x indicates a don’t care.
<i>x</i>	An italicized <i>x</i> indicates an alphanumeric variable
<i>n</i>	An italicized <i>n</i> indicates a numeric variable

- ¬ NOT logical operator
- & AND logical operator
- | OR logical operator

0 0 0 0

Indicates reserved bits or bit fields in a register. Although these bits may be written to as ones or zeros, they are always read as zeros.

Additional conventions used with instruction encodings are described in [Section A.2, “VLE Instruction Formats.”](#)

Acronyms and Abbreviations

[Table i](#) contains acronyms and abbreviations that are used in this document. Note that the meanings for some acronyms (such as XER) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
CR	Condition register
CTR	Count register
DEC	Decrementer register
EA	Effective address
EREF	Programmer's Reference Manual for Freescale Embedded Processors
GPR	General-purpose register
IEEE	Institute of Electrical and Electronics Engineers
IU	Integer unit
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
No-op	No operation
OEA	Operating environment architecture
PMC n	Performance monitor counter register
PVR	Processor version register
RISC	Reduced instruction set computing
RTL	Register transfer language

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
SIMM	Signed immediate value
SPR	Special-purpose register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
TB	Time base facility
TBL	Time base lower register
TBU	Time base upper register
TLB	Translation lookaside buffer
UIMM	Unsigned immediate value
UISA	User instruction set architecture
VA	Virtual address
VEA	Virtual environment architecture
VLEPEM	<i>VLE Programming Environments Manual</i>
VLEPIM	<i>VLE Technology Programming Interface Manual</i>
XER	Register used for indicating conditions such as carries and overflows for integer operations

Terminology Conventions

[Table ii](#) lists certain terms used in this manual that differ from the architecture terminology conventions.

Table ii. Terminology Conventions

The Architecture Specification	This Manual
Extended mnemonics	Simplified mnemonics
Fixed-point unit (FXU)	Integer unit (IU)
Privileged mode (or privileged state)	Supervisor-level privilege
Problem mode (or problem state)	User-level privilege
Real address	Physical address
Relocation	Translation
Storage (locations)	Memory
Storage (the act of)	Access
Store in	Write back
Store through	Write through

Table iii describes instruction field notation conventions used in this manual.

Table iii. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	crbA, crbB, crbD (respectively)
BF, BFA	crfD, crfS (respectively)
D	d
DS	ds
/, //, ///	0...0 (shaded)
RA, RB, RT, RS	rA, rB, rD, rS (respectively)
SI	SIMM
U	IMM
UI	UIMM



Chapter 1

Introduction

This chapter describes computation modes, document conventions, a processor overview, instruction formats, storage addressing, and instruction addressing for the variable length encoding (VLE) programming model.

1.1 Overview

VLE is a re-encoding of much of the Power instruction set using both 16- and 32-bit instruction formats. VLE is defined as a supplement to the Power ISA. Code pages using VLE encoding or non-VLE encoding can be intermingled in a system providing focus on both high performance and code density.

Offering 16-bit versions of Power instructions makes it possible to implement more space-efficient binary representations of applications for embedded environments where code density may affect overall system cost and, to a somewhat lesser extent, performance. This set of alternate encodings is selected on a page basis. A single storage attribute bit selects between standard instruction encodings and VLE instructions for that page.

Instruction encodings in pages marked as VLE are either 16 or 32 bits long and are aligned on 16-bit boundaries; therefore, all instruction pages marked as VLE must use big-endian byte ordering.

The programming model uses the same register set with both instruction set encodings, although some registers are not accessible by VLE instructions using the 16-bit formats and not all condition register (CR) fields are used by conditional branch instructions or instructions that access the CR executing from a VLE instruction page. In addition, due to the more restrictive encodings imposed by VLE instruction formats, immediate fields and displacements differ in size and use.

VLE additional instruction fields are described in the EREF.

Other than the requirement of big-endian byte ordering for instruction pages and the additional storage attribute to identify whether the instruction page corresponds to a VLE section of code, VLE complies with the memory model, register model, timer facilities, debug facilities, and interrupt/exception model defined in the user instruction set architecture (UISA), the virtual environment architecture (VEA), and the operating environment architecture (OEA). VLE instructions therefore execute in the same environment as non-VLE instructions.

1.2 Documentation Conventions

Book VLE adheres to the documentation conventions defined in the EREF. Note, however, that this book defines instructions that apply to the UISA, VEA, and OEA.

1.2.1 Description of Instruction Operation

The RTL (register transfer language) descriptions in Book VLE conform to the conventions described in the EREF.

1.3 Instruction Mnemonics and Operands

The description of each instruction includes the mnemonic and a formatted list of operands. VLE instruction semantics are either identical or similar to those of other instructions in the architecture, as described in the following:

- Where the semantics, side-effects, and binary encodings are identical, the standard mnemonics and formats are used. Such unchanged instructions are listed ([Table B-2](#) and [Table B-3](#)) and appropriately referenced, but the instruction definitions are not replicated in this book.
- Where the semantics are similar but the binary encodings differ, the standard mnemonic is typically preceded with an **e_** to denote a VLE instruction. To distinguish between similar instructions available in both 16- and 32-bit forms under VLE and standard instructions, VLE instructions encoded with 16 bits have an **se_** prefix.

Examples of VLE-supported instructions are shown below:

- **stwx rS,rA,rB**—Standard UISA instruction
- **e_stw rS,D(rA)**—32-bit VLE instruction
- **se_stw rZ,SD4(rX)**—16-bit VLE instruction

Chapter 2

Instruction Model

This chapter provides an overview of the VLE instruction model, including the following:

- [Section 2.1, “VLE Storage Addressing”](#)
- [Section 2.2, “VLE Compatibility with the Standard Architecture”](#)
- [Section 2.3, “Branch Operation Instructions”](#)
- [Section 2.4, “Integer Instructions”](#)
- [Section 2.5, “Storage Control Instructions”](#)
- [Section 2.6, “Additional Categories Available in VLE”](#)

2.1 VLE Storage Addressing

A program references memory using the effective address (EA) the processor computes when it executes a memory access or branch instruction (or certain other instructions defined in the VEA and OEA) or when it fetches the next sequential instruction.

2.1.1 Data Storage Addressing Modes

[Table 2-1](#) lists data storage addressing modes supported by VLE. Instruction forms are described in [Appendix A, “VLE Instruction Formats.”](#)

Table 2-1. Data Storage Addressing Modes

Mode	Description
Base+16-bit displacement (D-form, 32-bit format)	The 16-bit D field is sign-extended and added to the contents of the GPR designated by rA or to zero if rA = 0 to produce the EA.
Base+8-bit displacement (D8-form, 32-bit format)	The 8-bit D8 field is sign-extended and added to the contents of the GPR designated by rA or to zero if rA = 0 to produce the EA.
Base+scaled 4-bit displacement (SD4-form, 16-bit format)	The 4-bit SD4 field zero-extended, scaled (shifted left) according to the size of the operand, and added to the contents of the GPR designated by rX to produce the EA. (Note that rX = 0 is not a special case.)
Base+Index (X-form, 32-bit format)	The GPR contents designated by rB are added to the GPR contents designated by rA or to zero if rA = 0 to produce the EA.

2.1.2 Instruction Storage Addressing Modes

Table 2-2 lists instruction storage addressing modes supported by VLE.

Table 2-2. Instruction Storage Addressing Modes

Mode	Description
Taken BD24-form branch instructions (32-bit instruction format)	The 24-bit BD24 field is concatenated on the right with 0b0, sign-extended, and then added to the address of the branch instruction.
Taken B15-form branch instructions (32-bit instruction format)	The 15-bit BD15 field is concatenated on the right with 0b0, sign-extended, and then added to the address of the branch instruction to form the EA of the next instruction.
Take BD8-form branch instructions (16-bit instruction format)	The 8-bit BD8 field is concatenated on the right with 0b0, sign-extended, and then added to the address of the branch instruction to form the EA of the next instruction.
Sequential instruction fetching (or non-taken branch instructions)	The value 4 [2] is added to the address of the current 32-bit [16-bit] instruction to form the EA of the next instruction. If the address of the current instruction is 0xFFFF_FFFF_FFFF_FFFC [0xFFFF_FFFF_FFFF_FFFE] in 64-bit mode or 0xFFFF_FFFC [0xFFFF_FFFE] in 32-bit mode, the address of the next sequential instruction is undefined.
Any branch instruction with LK = 1 (32-bit instruction format)	The value 4 is added to the address of the current branch instruction and the result is placed into the LR. If the address of the current instruction is 0xFFFF_FFFF_FFFF_FFFC in 64-bit mode or 0xFFFF_FFFC in 32-bit mode, the result placed into the LR is undefined.
Branch se_bl , se_btrl , se_bctrl instructions (16-bit instruction format)	The value 2 is added to the address of the current branch instruction and the result is placed into the LR. If the address of the current instruction is 0xFFFF_FFFF_FFFF_FFFE in 64-bit mode or 0xFFFF_FFFE in 32-bit mode, the result placed into the LR is undefined.

2.1.2.1 Misaligned, Mismatched, and Byte-Ordering Instruction Storage Exceptions

A misaligned instruction storage exception occurs when an implementation that supports VLE attempts to execute an instruction that is not 32-bit aligned and the VLE storage attribute is not set for the page that corresponds to the effective address of the instruction. The attempted execution can be the result of a branch instruction that has bit 62 of the target address set or the result of an **rfi**, **se_rfi**, **rfci**, **se_rfci**, **rfdi**, **se_rfdi**, **rfmci**, or **se_rfmci** instruction that has bit 62 set in the respective save/restore register. If a misaligned instruction storage exception is detected and no higher priority exception exists, an instruction storage interrupt occurs, setting SRR0 to the misaligned address for which execution was attempted.

A mismatched instruction storage exception occurs when an implementation that supports VLE attempts to execute an instruction that crosses a page boundary for which the first page has the VLE storage attribute set and the second page has the VLE storage attribute bit cleared. If a mismatched instruction storage exception is detected and no higher priority exception exists, an instruction storage interrupt occurs, setting SRR0 to the misaligned address for which execution was attempted.

A byte-ordering instruction storage exception occurs when an implementation that supports VLE attempts to execute an instruction that has the VLE storage attribute set and the E (Endian) storage attribute set for the page that corresponds to the effective address of the instruction. If a byte-ordering instruction storage exception is detected and no higher priority exception exists, an instruction storage interrupt occurs, setting SRR0 to the address for which execution was attempted.

2.1.2.2 VLE Exception Syndrome Bits

VLE defines the following bits to facilitate VLE exception handling, as described in the EREF:

- ESR[VLEMI] is set when an exception and subsequent interrupt is caused by the execution or attempted execution of an instruction that resides in memory with the VLE storage attribute set.
- ESR[MIF] is set when an instruction storage interrupt is caused by a misaligned instruction storage exception, or when an instruction TLB error interrupt was caused by a TLB miss on the second half of a misaligned 32-bit instruction.
- ESR[BO] is set when an instruction storage interrupt is caused by a mismatched instruction storage exception or a byte-ordering instruction storage exception.

NOTE (Programming)

When an instruction TLB error interrupt occurs as the result of an instruction TLB miss on the second half of a 32-bit VLE instruction that is aligned to only 16-bits, SRR0 points to the first half of the instruction and ESR[MIF] is set. Any other status posted as a result of the TLB miss (such as MAS register updates) reflects the page corresponding to the second half of the instruction that caused the instruction TLB miss.

2.2 VLE Compatibility with the Standard Architecture

This chapter addresses the relationship between VLE and the standard architecture.

2.2.1 Overview

VLE uses the same semantics as other Power ISA instructions. Due to the limited instruction encoding formats, VLE instructions typically support reduced immediate fields and displacements, and not all operations defined by the standard architecture are encoded in VLE. The design criteria are to capture all useful operations, with most frequent operations given priority. Immediate fields and displacements are provided to cover the majority of ranges encountered in embedded control code. Instructions are encoded in either a 16- or 32-bit format, and these may be freely intermixed.

VLE instructions cannot access FPRs. VLE instructions use GPRs and SPRs with the following limitations:

- Most VLE instructions using 16-bit formats are limited to addressing GPR0–GPR7 and GPR24–GPR31. Move instructions are provided to transfer register contents between these registers and GPR8–GPR23.
- VLE compare and bit test instructions using the 16-bit formats implicitly set their results in CR0.

VLE instruction encodings are generally different than instructions defined by the standard architecture, except that most instructions falling within primary opcode 31 are encoded identically and have identical semantics unless they affect or access a resource unsupported by VLE.

2.2.2 VLE Processor and Storage Control Extensions

This section describes additional functionality to support VLE.

2.2.2.1 Instruction Extensions

This section describes extensions to support VLE operations. Because instructions may reside on a half-word boundary, bit 62 is not masked by instructions that read an instruction address from a register, such as LR, CTR, or a save/restore register 0 that holds an instruction address:

The instruction set defined by the standard architecture is modified to support halfword instruction addressing, as follows:

- Return from interrupt instructions (**rfi**, **rfdi**, **rfdi**, and **rfmci**) no longer mask bit 62 of the respective save/restore register 0. The destination address is $SRR0[0-62] \parallel 0b0$, $CSRR0[0-62] \parallel 0b0$, $DSRR0[0-62] \parallel 0b0$, $MCSR0[0-62] \parallel 0b0$, respectively.
- **bclr**, **bclrl**, **bcctr**, and **bcctrl** no longer mask bit 62 of LR or CTR. The destination address is $LR[0-62] \parallel 0b0$ or $CTR[0-62] \parallel 0b0$.

2.2.2.2 MMU Extensions

VLE operation is indicated by the VLE storage page attribute. When this attribute is set, instruction fetches from that page are decoded and processed as VLE instructions. See the EREF.

When instructions execute from a page whose VLE storage attribute is set, the processor is in VLE mode.

2.2.3 VLE Limitations

VLE instruction fetches are valid only when performed in a big-endian mode. Attempting to fetch an instruction in a little-endian mode from a page with the VLE storage attribute set causes an instruction storage byte-ordering exception.

Support for concurrent modification and execution of VLE instructions is implementation-dependent.

2.3 Branch Operation Instructions

This section describes branch instructions that can be executed when a processor is in VLE mode. It also describes the registers that support them.

2.3.1 Branch Processor Registers

The following registers support branch operations:

- [Section 2.3.1.1, “Condition Register \(CR\)”](#)
- [Section 2.3.1.2, “Link Register \(LR\)”](#)
- [Section 2.3.1.3, “Count Register \(CTR\)”](#)

2.3.1.1 Condition Register (CR)

The CR reflects the result of certain operations and provides a mechanism for testing (and branching).

VLE uses the entire CR, as described in the EREF, but some comparison operations and all branch instructions are limited to using CR0–CR3. The full UISA-defined CR field and logical operations are provided, however.

CR bits are grouped into eight 4-bit fields, CR field 0 (CR0) ... CR field 7 (CR7), which are set by VLE-defined instructions in one of the following ways:

- Specified CR fields can be set by a move to the CR from a GPR (**mtrcf**, **mtocrf**).
- A specified CR field can be set by a move to the CR from another CR field (**e_mcrf**) or from XER[32–35] (**mcrxr**).
- CR field 0 can be set as the implicit result of an integer instruction.
- A specified CR field can be set as the result of an integer compare instruction.
- CR field 0 can be set as the result of an integer bit test instruction.

Other instructions from implemented categories may also set bits in the CR in the same manner that they would when not in VLE mode.

Instructions are provided to perform logical operations on individual CR bits and to test individual CR bits.

For all integer instructions in which the Rc bit is defined and set, and for **e_add2i.**, **e_and2i.**, and **e_and2is.**, the first three bits of CR field 0 (CR_{32:34}) are set by signed comparison of the result to zero, and the fourth bit of CR field 0 (CR[35]) is copied from the final state of XER[SO]. “Result” here refers to the entire 64-bit value placed into the target register in 64-bit mode, and to bits 32–63 of the value placed into the target register in 32-bit mode.

```

if (64-bit mode)
  then M ← 0
  else M ← 32
if (target_register)M:63 < 0 then c ← 0b100
else if (target_register)M:63 > 0 then c ← 0b010
else c ← 0b001
CR0 ← c || XERSO

```

If any portion of the result is undefined, the value placed into the first three bits of CR field 0 is undefined.

The bits of CR field 0 are interpreted as shown in [Table 2-3](#).

Table 2-3. CR0 Field Descriptions

Bits	Name	Description
32	LT	Negative—The result is negative.
33	GT	Positive—The result is positive.
34	EQ	Zero—The result is 0.
35	SO	Summary overflow—This is a copy of the contents of XER[SO] at the completion of the instruction.

2.3.1.1.1 Condition Register Setting for Compare Instructions

For compare instructions, a CR field specified by the BF operand for the **e_cmpph**, **e_cmpphl**, **e_cmpi**, and **e_cmpli** instructions, or CR0 for the **se_cmpl**, **e_cmp16i**, **e_cmph16i**, **e_cmphl16i**, **e_cmpl16i**, **se_cmp**, **se_cmpph**, **se_cmphl**, **se_cmpi**, and **se_cmpli** instructions, is set to reflect the result of the comparison. The CR field bits are interpreted as shown below. A complete description of how the bits are set is given in the instruction descriptions and the EREF.

Condition register bits settings for compare instructions are interpreted as shown in [Table 2-4](#).

Table 2-4. Condition Register Settings for Compare Instructions

CR Bit ¹	Description
4×BF + 32	Less than (LT). For signed integer compare, (rA) or (rX) < sci8, SI, (rB), or (rY). For unsigned integer compare, (rA) or (rX) < ^u sci8, UI, UI5, (rB), or (rY).
4×BF + 33	Greater than (GT). For signed integer compare, (rA) or (rX) > sci8, SI, (rB), or (rY). For unsigned integer compare, (rA) or (rX) > ^u sci8, UI, UI5, (rB), or (rY).
4×BF + 34	Equal (EQ). For integer compare, (rA) or (rX) = sci8, UI, UI5, SI, (rB), or (rY).
4×BF + 35	Summary overflow (SO). For integer compare, this is a copy of XER[SO] at the completion of the instruction.

¹ **e_cmpi**, and **e_cmpli** instructions have a BF32 field instead of a BF field; for these instructions, BF32 should be substituted for BF in the table.

2.3.1.1.2 Condition Register Setting for the Bit Test Instruction

The Bit Test Immediate instruction, **se_btsti**, also sets CR field 0. See the instruction description and also the EREF.

2.3.1.2 Link Register (LR)

VLE instructions use the LR as defined in the UISA, although VLE defines a subset of the variants defined for conditional branches involving the LR.

2.3.1.3 Count Register (CTR)

VLE instructions use the CTR as defined in the UISA, although VLE defines a subset of the variants defined for conditional branches involving the CTR.

2.3.2 Branch Instructions

The sequence of instruction execution can be changed by the branch instructions. Because VLE instructions must be aligned on half-word boundaries, the low-order bit of the generated branch target address is forced to 0 by the processor in performing the branch.

The branch instructions compute the EA of the target in one of the following ways, as described in [Section 2.1.2, “Instruction Storage Addressing Modes.”](#)

1. Adding a displacement to the address of the branch instruction.
2. Using the address contained in the LR (Branch to Link Register [and Link]).

3. Using the address contained in the CTR (Branch to Count Register [and Link]).

Branching can be conditional or unconditional, and the return address can optionally be provided. If the return address is to be provided ($LK = 1$), the EA of the instruction following the branch instruction is placed into the LR after the branch target address has been computed; this is done regardless of whether the branch is taken.

In branch conditional instructions, the BI32 or BI16 instruction field specifies the CR bit to be tested. For 32-bit instructions using BI32, CR[32–47] (corresponding to bits in CR0:CR3) may be specified. For 16-bit instructions using BI16, only CR[32–35] (bits within CR0) may be specified.

In branch conditional instructions, the BO32 or BO16 field specifies the conditions under which the branch is taken and how the branch is affected by or affects the CR and CTR. Note that VLE instructions also have different encodings for the BO32 and BO16 fields than in the UISA-defined BO field.

If the BO32 field specifies that the CTR is to be decremented, in 64-bit mode CTR[0–63] are decremented, and in 32-bit mode CTR[32–63] are decremented. If BO16 or BO32 specifies a condition that must be TRUE or FALSE, that condition is obtained from the contents of CR[BI32+32] or CR[BI16+32]. (Note that CR bits are numbered 32–63. BI32 or BI16 refers to the condition register bit field in the branch instruction encoding. For example, specifying BI32 = 2 refers to CR[34].)

For [Table 2-5](#), let $M = 0$ in 64-bit mode and $M = 32$ in 32-bit mode.

Encodings for the BO32 field for VLE are shown in [Table 2-5](#).

Table 2-5. BO32 Field Encodings

BO32	Description
00	Branch if the condition is false.
01	Branch if the condition is true.
10	Decrement CTR[M–63], then branch if the decremented value $\neq 0$
11	Decrement CTR[M–63], then branch if the decremented value = 0.

Encodings for the BO16 field for VLE are shown in [Table 2-6](#).

Table 2-6. BO16 Field Encodings

BO16	Description
0	Branch if the condition is false.
1	Branch if the condition is true.

2.3.3 System Linkage Instructions

The system linkage instructions enable the program to call on the system to perform a service and provide a means by which the system can return from performing a service or from processing an interrupt. System linkage instructions defined by VLE are identical in semantics to system linkage instructions defined in the UISA and OEA, with the exception of the LEV field, but are encoded differently.

se_sc provides the same functionality as **sc** without the LEV field. **se_rfi**, **se_rfci**, **se_rfdi**, and **se_rfmci** provide the same functionality as **rfi**, **rfci**, **rfdi**, and **rfmci**, respectively.

2.4 Integer Instructions

This section lists the integer instructions supported by VLE.

2.4.1 Integer Load Instructions

The integer load instructions compute the EA of the memory to be accessed as described in [Section 2.1.1, “Data Storage Addressing Modes.”](#)

The byte, halfword, word, or doubleword in storage addressed by EA is loaded into **rD** or **rZ**.

VLE supports both big- and little-endian byte ordering for data accesses.

Some integer load instructions have an update form in which **rA** is updated with the EA. For these forms, if **rA** ≠ 0 and **rA** ≠ **rD**, the EA is placed into **rA** and the memory element (byte, halfword, word, or doubleword) addressed by EA is loaded into **rD**. If **rA** = 0 or **rA** = **rD**, the instruction form is invalid. This is the same behavior as specified for UISA load with update instructions.

The integer load instructions, **lbzx**, **lbzux**, **lhzx**, **lhzux**, **lwzx**, and **lwzux**, are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those defined by the UISA. See the EREF for the instruction definitions.

The integer load instructions, **lwax**, **lwaux**, **ldx**, and **ldux**, are available in VLE mode on 64-bit implementations. The mnemonics, decoding, and semantics for these instructions are identical to those in the UISA. See the EREF for the instruction definitions.

2.4.2 Integer Load and Store with Byte Reversal Instructions

The integer load with byte reversal and store with byte reversal instructions, **lhbrx**, **lwbrx**, **sthbrx**, and **stwbrx**, are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those in the UISA. See the EREF for instruction definitions.

2.4.3 Integer Load and Store Multiple Instructions

The load/store multiple instructions have preferred forms; see the EREF. In the preferred forms storage alignment satisfies the following rule.

- The combination of the EA and **rD** (**rS**) is such that the low-order byte of GPR 31 is loaded (stored) from (into) the last byte of an aligned quadword in storage.

2.4.4 Integer Arithmetic Instructions

The integer arithmetic instructions use the contents of the GPRs as source operands and place results into GPRs, into status bits in the XER, and into CR0.

The integer arithmetic instructions treat source operands as signed integers unless an instruction is explicitly identified as performing an unsigned operation.

The **e_and2i**. instruction and other arithmetic instructions with Rc=1 set the first three bits of CR0 to characterize the result placed into the target register. In 64-bit mode, these bits are set by signed comparison of the result to 0. In 32-bit mode, these bits are set by signed comparison of the low-order 32 bits of the result to zero.

e_addic[,] and **e_subfic**[,] always set CA to reflect the carry out of bit 0 in 64-bit mode and out of bit 32 in 32-bit mode.

The integer arithmetic instructions, **add**[,] , **addo**[,] , **addc**[,] , **addco**[,] , **adde**[,] , **addeo**[,] , **addme**[,] , **addmeo**[,] , **addze**[,] , **addzeo**[,] , **divw**[,] , **divwo**[,] , **divwu**[,] , **divwuo**[,] , **mulhw**[,] , **mulhwu**[,] , **mullw**[,] , **mullwo**[,] , **neg**[,] , **nego**[,] , **subf**[,] , **subfo**[,] , **subfe**[,] , **subfeo**[,] , **subfme**[,] , **subfmeo**[,] , **subfze**[,] , **subfzeo**[,] , **subfc**[,] , and **subfco**[,] are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those in the UISA; see the EREF for the instruction definitions.

The integer arithmetic instructions, **mulld**[,] , **mulldo**[,] , **mulhd**[,] , **muldu**[,] , **divd**[,] , **divdo**[,] , **divdu**[,] , and **divduo**[,] are available in VLE mode on 64-bit implementations. The mnemonics, decoding, and semantics for those instructions are identical to these in the UISA; see the EREF for the instruction definitions.

2.4.5 Integer Trap Instructions

The integer trap instruction **tw** is available in VLE mode. The mnemonics, decoding, and semantics for this instruction are identical to that in the UISA; see the EREF for the instruction definition.

The integer trap instruction **td** is available in VLE mode on 64-bit implementations. The mnemonic, decoding, and semantics for the **td** instruction are identical to those in the UISA; see the EREF for the instruction definitions.

2.4.6 Integer Select Instruction

The Integer Select instruction **isel** provides a means to select one of two registers and place the result in a destination register under the control of a predicate value supplied by a CR bit.

The **isel** is available in VLE mode. The mnemonics, decoding, and semantics for this instruction are identical to that in the UISA; see the EREF for the instruction definition.

2.4.7 Integer Logical, Bit, and Move Instructions

The logical instructions perform bit-parallel operations on 64-bit operands. The bit instructions manipulate a bit, or create a bit mask, in a register. The move instructions move a register or an immediate value into a register.

The X-form logical instructions with Rc=1, the SCI8-form logical instructions with Rc=1, the RR-form logical instructions with Rc=1, the **e_and2i**. instruction, and the **e_and2is**. instruction set the first three bits of CR field 0 as the arithmetic instructions described in [Section 2.4.4, “Integer Arithmetic](#)

Instructions.” (Also see [Section 2.3.1.1, “Condition Register \(CR\).”](#)) The logical instructions do not change the SO, OV, and CA bits in the XER.

The integer logical instructions, **and[.]**, **or[.]**, **xor[.]**, **nand[.]**, **nor[.]**, **eqv[.]**, **andc[.]**, **orc[.]**, **extsb[.]**, **extsh[.]**, **cntlzw[.]**, and **popcntb**, are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those in the UISA; see the EREF for instruction definitions.

The integer logical instructions, **extsw[.]** and **cntlzd[.]**, are available in VLE mode on 64-bit implementations. The mnemonics, decoding, and semantics for these instructions are identical to those in the UISA; see the EREF for instruction definitions.

2.5 Storage Control Instructions

2.5.1 Storage Synchronization Instructions

The memory synchronization instructions implemented by VLE are identical in semantics to those defined in the VEA and OEA. The **se_isync** instruction is defined by VLE but has the same semantics as **isync**.

The load and reserve instruction **lwarx** and the store conditional instruction **stwcx** are available in VLE mode. The mnemonics, decoding, and semantics for those instructions are identical to those in the VEA; see the EREF for instruction definitions.

The load and reserve instruction **ldarx** and the store conditional instruction **stdcx** are available in VLE mode on 64-bit implementations. The mnemonics, decoding, and semantics for those instructions are identical to those in the VEA; see the EREF for instruction definitions.

Memory barrier instructions, **sync (msync)** and **mbar** are available in VLE mode. The mnemonics, decoding, and semantics for those instructions are identical to those in the VEA; see the EREF for instruction definitions.

The **wait** instruction is available in VLE mode if the category Wait is implemented. The mnemonics, decoding, and semantics for **wait** are identical to those in the VEA; see the EREF for the instruction definition.

2.5.2 Cache Management Instructions

Cache management instructions implemented by VLE are identical to those defined in the VEA and OEA.

The cache management instructions, **dcba**, **dcbf**, **dcbst**, **dcbt**, **dcbstst**, **dcbz**, **icbi**, and **icbt**, are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those in the VEA; see the EREF instruction definitions.

dcbi is available in VLE mode. The mnemonics, decoding, and semantics for this instruction are identical to those in the OEA; see the EREF for instruction definition.

2.5.3 Cache Locking Instructions

Cache locking instructions implemented by VLE are identical to those defined by the OEA. If the cache locking instructions are implemented in VLE, the embedded cache locking instructions must also be implemented.

The cache locking instructions defined by the OEA, **dcbtls**, **dcbtstls**, **dcblc**, **icbtls**, and **icblc** are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those defined in the OEA; see the EREF for instruction definitions.

2.5.4 TLB Management Instructions

The TLB management instructions implemented by VLE are identical to those defined by the OEA.

The OEA-defined TLB management instructions, **tlbre**, **tlbwe**, **tlbivax**, **tlbsync**, and **tlbsx**, are available in VLE mode. The mnemonics, decoding, and semantics for these instructions are identical to those in the OEA. See the EREF for instruction definitions.

Instructions and resources from category Embedded.MMU Type FSL are available if that category is implemented.

2.5.5 Instruction Alignment and Byte Ordering

Only big-endian instruction memory is supported when executing from a page of VLE instructions. Attempting to fetch VLE instructions from a page marked as little-endian generates an instruction storage interrupt byte-ordering exception.

2.6 Additional Categories Available in VLE

Processors that implement VLE may implement instructions and resources from categories other than Base and Embedded. User documentation for each processor core indicates which of these may be implemented. These instructions are described in the EREF.

Such categories include those for which all the instructions in the category use primary opcode 4 or primary opcode 31, as listed below:

- **Move assist.** Move assist instructions implemented by VLE are identical to those defined in the UISA. The mnemonics, decoding, and semantics for those instructions are identical to those in the UISA; see the EREF for the instruction definitions.
- **Vector.** Vector instructions implemented by VLE are identical to those defined in the UISA. The mnemonics, decoding, and semantics for those instructions are identical to those in the UISA; see the EREF for the instruction definitions.
- **Signal processing engine (SPE).** SPE instructions implemented by VLE are identical to those defined in the UISA. The mnemonics, decoding, and semantics for those instructions are identical to those in the UISA; see the EREF for the instruction definitions.
- **Embedded floating-point.** Embedded floating-point instructions implemented by VLE are identical to those defined in the UISA. The mnemonics, decoding, and semantics for those instructions are identical to those in the UISA; see the EREF for the instruction definitions.

- Legacy move assist. Legacy move assist instructions implemented by VLE are identical to those defined in the UISA. The mnemonics, decoding, and semantics for those instructions are identical to those in the UISA; see the EREF for instruction definitions.
- External PID. External process ID instructions implemented by VLE are identical to those defined by the OEA. Semantics for those instructions are identical to those in the OEA; see the EREF for the instruction definitions.
- Embedded performance monitor. Embedded performance monitor instructions implemented by VLE are identical to those defined by the OEA. The mnemonics, decoding, and semantics for those instructions are identical to those in the OEA; see the EREF for the instruction definitions.
- Processor control. Processor control instructions implemented by VLE are identical to those defined by the OEA. The mnemonics, decoding, and semantics for those instructions are identical to those in the OEA; see the EREF for the instruction definitions.

Chapter 3

VLE Instruction Set

The VLE extension ISA is defined in the instruction pages in this chapter. Because of the various immediate field and displacement field calculations used in the VLE extension, a description of the less obvious ones precedes the actual instruction pages, and the instruction descriptions generally assume the appropriate calculation has been performed.

NOTE

The instructions in this section are listed in order of the root instruction. For example, `e_cmpi` and `se_cmpi` are both listed under `cmpi`.

3.1 Supported Power ISA Instructions

Table 3-1 lists instructions that are used by the VLE extension that are defined by the UISA, VEA, or OEA. Those instructions are described in the EREF.

Descriptions in this chapter indicate any limitations on the behavior of VLE instructions as compared to their non-VLE equivalents.

Table 3-1. Non-VLE Instructions Listed by Mnemonic

Mnemonic	Instruction
<code>add rD,rA,rB</code> <code>add. rD,rA,rB</code> <code>addo rD,rA,rB</code> <code>addo. rD,rA,rB</code>	Add
<code>addc rD,rA,rB</code> <code>addc. rD,rA,rB</code> <code>addco rD,rA,rB</code> <code>addco. rD,rA,rB</code>	Add Carrying
<code>adde rD,rA,rB</code> <code>adde. rD,rA,rB</code> <code>addeo rD,rA,rB</code> <code>addeo. rD,rA,rB</code>	Add Extended
<code>andc[.] rA,rS,rB</code>	AND with Complement
<code>and[.] rA,rS,rB</code>	AND
<code>cmp crD,L,rA,rB</code>	Compare
<code>cmpl crD,L,rA,rB</code>	Compare Logical
<code>cntlzw rA,rS</code> <code>cntlzw. rA,rS</code>	Count Leading Zeros Word
<code>dcba rA,rB</code>	Data Cache Block Allocate

Table 3-1. Non-VLE Instructions Listed by Mnemonic (continued)

Mnemonic	Instruction
dcbf rA,rB	Data Cache Block Flush
dcbi rA,rB	Data Cache Block Invalidate
dcbst rA,rB	Data Cache Block Store
dcbt CT,rA,rB	Data Cache Block Touch
dcbtst CT,rA,rB	Data Cache Block Touch for Store
dcbz rA,rB	Data Cache Block set to Zero
divw rD,rA,rB divw. rD,rA,rB divwo rD,rA,rB divwo. rD,rA,rB	Divide Word
divwu rD,rA,rB divwu. rD,rA,rB divwuo rD,rA,rB divwuo. rD,rA,rB	Divide Word Unsigned
eqv rA,rS,rB eqv. rA,rS,rB	Equivalent
extsb rA,rS extsb. rA,rS	Extend Sign Byte
extsh rA,rS extsh. rA,rS	Extend Sign Halfword
e_srwi rA,rS,SH	Shift Right Word Immediate
icbi rA,rB	Instruction Cache Block Invalidate
icbt CT,rA,rB	Instruction Cache Block Touch
lbzx rD,rA,rB lbzux rD,rA,rB	Load Byte and Zero Indexed Load Byte and Zero with Update Indexed
lhax rD,rA,rB lhaux rD,rA,rB	Load Halfword Algebraic Indexed Load Halfword Algebraic with Update Indexed
lhbrx rD,rA,rB	Load Halfword Byte-Reverse Indexed
lhzx rD,rA,rB lhzux rD,rA,rB	Load Halfword and Zero Indexed Load Halfword and Zero with Update Indexed
lwarx rD,rA,rB	Load Word And Reserve Indexed
lwbrx rD,rA,rB	Load Word Byte-Reverse Indexed
lwzx rD,rA,rB lwzux rD,rA,rB	Load Word and Zero Indexed Load Word and Zero with Update Indexed
mbar	Memory Barrier
mcrxr crD	Move to Condition Register from Integer Exception Register
mfcrr rD	Move From condition register
mfdcr rD,DCRN	Move From Device Control Register

Table 3-1. Non-VLE Instructions Listed by Mnemonic (continued)

Mnemonic	Instruction
mfmsr rD	Move From Machine State Register
mfspir rD,SPRN	Move From Special Purpose Register
msync	Memory Synchronize
mtrcf FXM,rS	Move to Condition Register Fields
mtocr DCRN,rS	Move To Device Control Register
mtmsr rS	Move To Machine State Register
mtspir SPRN,rS	Move To Special Purpose Register
mulhw rD,rA,rB mulhw. rD,rA,rB	Multiply High Word
mulhwu rD,rA,rB mulhwu. rD,rA,rB	Multiply High Word Unsigned
mullw rD,rA,rB mullw. rD,rA,rB mullwo rD,rA,rB mullwo. rD,rA,rB	Multiply Low Word
nand rA,rS,rB nand. rA,rS,rB	NAND
neg rD,rA neg. rD,rA nego rD,rA nego. rD,rA	Negate
nor rA,rS,rB nor. rA,rS,rB	NOR
or rA,rS,rB or. rA,rS,rB	OR
orc rA,rS,rB orc. rA,rS,rB	OR with Complement
slw rA,rS,rB slw. rA,rS,rB	Shift Left Word
sraw rA,rS,rB sraw. rA,rS,rB	Shift Right Algebraic Word
srawi rA,rS,SH srawi. rA,rS,SH	Shift Right Algebraic Word Immediate
srw rA,rS,rB srw. rA,rS,rB	Shift Right Word
stbx rS,rA,rB stbux rS,rA,rB	Store Byte Indexed Store Byte with Update Indexed
sthbrx rS,rA,rB	Store Halfword Byte-Reverse Indexed
sthx rS,rA,rB sthux rS,rA,rB	Store Halfword Indexed Store Halfword with Update Indexed

Table 3-1. Non-VLE Instructions Listed by Mnemonic (continued)

Mnemonic	Instruction
stwbrx rS,rA,rB	Store Word Byte-Reverse Indexed
stwcx. rS,rA,rB	Store Word Conditional Indexed
stwx rS,rA,rB stwux rS,rA,rB	Store Word Indexed Store Word with Update Indexed
subf rD,rA,rB subf. rD,rA,rB subfo rD,rA,rB subfo. rD,rA,rB	Subtract From
subfc rD,rA,rB subfc. rD,rA,rB subfco rD,rA,rB subfco. rD,rA,rB	Subtract From Carrying
tlbivax rA,rB	TLB Invalidate Virtual Address Indexed
tlbre	TLB Read Entry
tlbsx rA,rB	TLB Search Indexed
tlbsync	TLB Synchronize
tlbwe	TLB Write Entry
tw TO,rA,rB	Trap Word
wrtee rA	Write MSR External Enable
wrteei E	Write MSR External Enable Immediate
xor rA,rS,rB xor. rA,rS,rB	XOR
isel rD,rA,rB,crb	Integer Select

3.2 Immediate Field and Displacement Field Encodings

Table 3-2 shows encodings for immediate and displacement fields.

Table 3-2. Immediate Field and Displacement Field Encodings

Encoding	Description
BD15	Format used by 32-bit branch conditional class instructions. The BD15 field is an 15-bit signed displacement which is sign-extended and shifted left one bit (concatenated with 0b0) and then added to the current instruction address to form the branch target address.
BD24	Format used by 32-bit branch class instructions. The BD24 field is an 24-bit signed displacement which is sign-extended and shifted left one bit (concatenated with 0b0) and then added to the current instruction address to form the branch target address.
BD8	Format used by 16-bit branch and branch conditional class instructions. The BD8 field is an 8-bit signed displacement which is sign-extended and shifted left one bit (concatenated with 0b0) and then added to the current instruction address to form the branch target address.

Table 3-2. Immediate Field and Displacement Field Encodings (continued)

Encoding	Description
D	Format used by some 32-bit load and store class instructions. The D field is a 16-bit signed displacement which is sign-extended to 32 bits, and then added to the base register to form a 32-bit EA.
D8	Format used by some 32-bit load and store class instructions. The D8 field is a 8-bit signed displacement which is sign-extended to 32 bits, and then added to the base register to form a 32-bit EA.
F, SCL, UI8 (SCI8 format)	Format used by some 32-bit arithmetic, compare, and logical instructions. The UI8 field is an 8-bit immediate value shifted left 0, 1, 2, or 3 byte positions according to the value of the SCL field. The remaining bits in the 32-bit word are filled with the value of the F field, and the resulting 32-bit value is used as one operand of the instruction. More formally, <pre> if SCL=0 then imm_value ← 24F UI8 else if SCL=1 then imm_value ← 16F UI8 8F else if SCL=2 then imm_value ← 8F UI8 16F else imm_value ← UI8 24F </pre>
LI20	Format used by 32-bit e_li instruction. The LI20 field is a 20-bit signed displacement which is sign-extended to 32 bits for the e_li instruction.
OIM5	Format used by the 16-bit se_addi , se_cmpli , and se_subi [.] instructions. The OIM5 instruction field is a 5-bit value in the range 0–31 and is used to represent immediate values in the range 1–32; thus the binary encoding of 0b00000 represents an immediate value of 1, 0b00001 represents an immediate value of 2, and so on. In the instruction descriptions, OIMM represents the immediate value, not the OIM5 instruction field binary encoding.
SCI8 format	Refer to F, SCL, UI8 (SCI8 format)
SD4	Format used by 16-bit load and store class instructions. The SD4 field is a 4-bit unsigned immediate value zero-extended to 32 bits, shifted left according to the size of the operation, and then added to the base register to form a 32-bit EA. For byte operations, no shift is performed. For half-word operations, the immediate is shifted left one bit (concatenated with 0b0). For word operations, the immediate is shifted left two bits (concatenated with 0b00). For future double-word operations, the immediate is shifted left three bits (concatenated with 0b000).
SI (D format, I16A format)	Format used by certain 32-bit arithmetic type instructions. The SI field is a 16-bit signed immediate value sign-extended to 32 bits and used as one operand of the instruction. The instruction encoding differs between the D and I16A instruction formats as shown in Figure A-11 and Figure A-12
UI (I16A, I16L formats)	Format used by certain 32-bit logical and arithmetic type instructions. The UI field is a 16-bit unsigned immediate value zero-extended to 32 bits or padded with 16 zeros and used as one operand of the instruction. The instruction encoding differs between the I16A and I16L instruction formats as shown in Figure A-12 and Figure A-13 .
UI5	This format is used by some 16-bit Reg+Imm class instructions. The UI5 field is a 5-bit unsigned immediate value zero-extended to 32 bits and used as the second operand of the instruction. For other 16-bit Reg+Imm class instructions, the UI5 field is a 5-bit unsigned immediate value used to select a register bit in the range 0–31.
UI7	This format is used by the 16-bit se_li instructions. The UI7 field is a 7-bit unsigned immediate value zero-extended to 32 bits and used as the operand of the instruction.

_addx

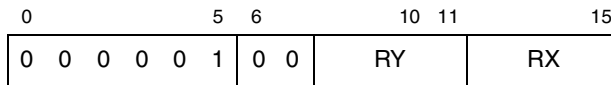
VLE	User
-----	------

_addx

Add

se_add

rX,rY



$$\text{sum}_{32:63} \leftarrow \text{GPR}(\text{RX}) + \text{GPR}(\text{RY})$$

$$\text{GPR}(\text{RX}) \leftarrow \text{sum}_{32:63}$$

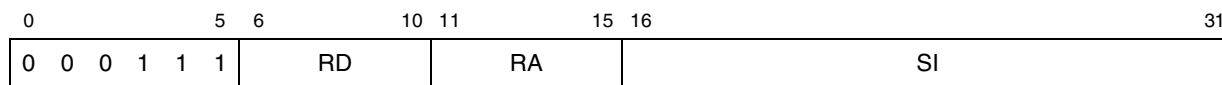
The sum of the contents of GPR(rX) and the contents of GPR(rY) is placed into GPR(rX).
 Special Registers Altered: None

_addix

VLE	User
-----	------

_addix

Add [2 operand] Immediate [Shifted] [and Record]

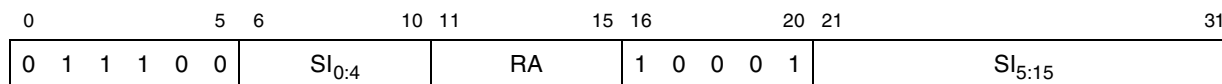
e_add16i **rD,rA,SI**

```

a ← GPR(RA)
b ← EXTS(SI)
GPR(RD) ← a + b

```

The sum of the contents of GPR(**rA**) and the sign-extended value of field **SI** is placed into GPR(**rD**).
Special Registers Altered: None

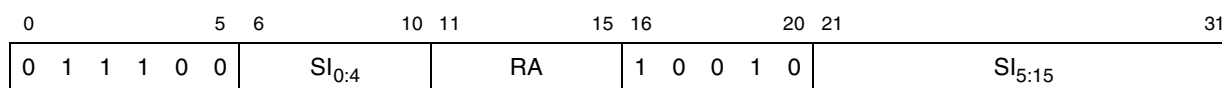
e_add2i. **rA,SI**

```

SI ← SI0:4 || SI5:15
sum32:63 ← GPR(RA) + EXTS(SI)
LT ← sum32:63 < 0
GT ← sum32:63 > 0
EQ ← sum32:63 = 0
CR0 ← LT || GT || EQ || SO
GPR(RA) ← sum32:63

```

The sum of the contents of GPR(**rA**) and the sign-extended value of **SI** is placed into GPR(**rA**).
Special Registers Altered: CR0

e_add2is **rA,SI**

```

SI ← SI0:4 || SI5:15
sum32:63 ← GPR(RD) + (SI || 160)
GPR(RA) ← sum32:63

```

The sum of the contents of GPR(**rA**) and the value of **SI** concatenated with 16 zeros is placed into GPR(**rA**).

Special Registers Altered: None

e_addi **rD,rA,SCI8****(Rc = 0)**

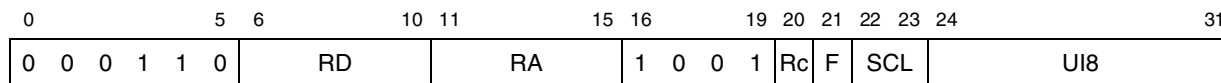
_addicx

VLE	User
-----	------

_addicx

Add Immediate Carrying [and Record]

e_addic	rD,rA,SCI8	(Rc = 0)
e_addic.	rD,rA,SCI8	(Rc = 1)



```

imm ← SCI8(F,SCL,UI8)
carry32:63 ← Carry(GPR(RA) + imm)
sum32:63 ← GPR(RA) + imm
if Rc=1 then do
    LT ← sum32:63 < 0
    GT ← sum32:63 > 0
    EQ ← sum32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RD) ← sum32:63
CA ← carry32

```

The sum of the contents of GPR(**rA**) and the value of SCI8 is placed into GPR(**rD**).

Special Registers Altered: CA, CR0 (if Rc=1)

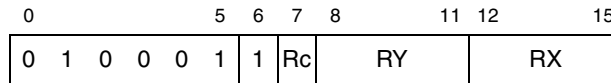
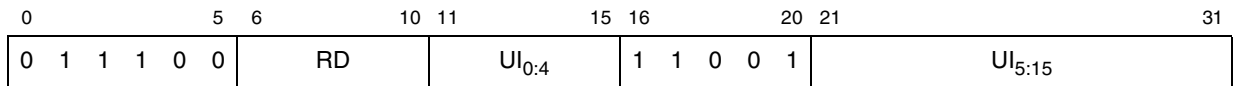
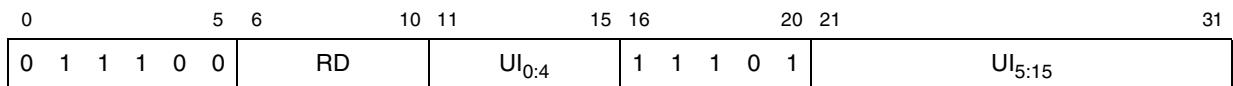
_andx

VLE	User
-----	------

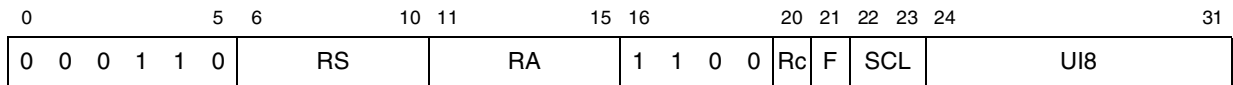
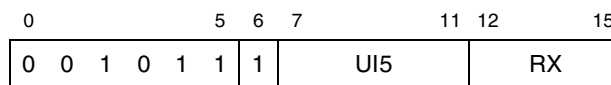
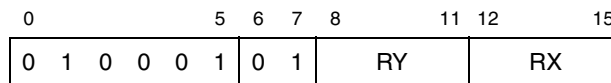
_andx

AND [2 operand] [Immediate I with Complement] [and Record]

se_and **rX,rY** (Rc = 0)
se_and. **rX,rY** (Rc = 1)

**e_and2i.** **rD,UI****e_and2is.** **rD,UI**

e_andi **rA,rS,SCI8** (Rc = 0)
e_andi. **rA,rS,SCI8** (Rc = 1)

**se_andi** **rX,UI5****se_andc** **rX,rY**

```

if 'e_andi[.]' then b ← SCI8(F,SCL,UI8)
if 'se_andi' then b ← UI5
if 'se_and[.]' then b ← GPR(RY)
if 'se_andc' then b ← ¬GPR(RY)
if 'e_and2i.' then b ← 160 || UI0:4 || UI5:15
if 'e_and2is.' then b ← UI0:4 || UI5:15 || 160
result32:63 ← GPR(RS or RD or RX) & b
if Rc=1 then do
    LT ← result32:63 < 0

```

```

GT ← result32:63 > 0
EQ ← result32:63 = 0
CR0 ← LT || GT || EQ || SO
if 'se_and[ci]' then GPR(RX) ← result32:63 else GPR(RA or RD) ←
result32:63

```

For **e_andi[.]**, the contents of GPR(**rS**) are ANDed with the value of SCI8.

For **e_and2i.**, the contents of GPR(**rD**) are ANDed with ¹⁶0 || UI.

For **e_and2is.**, the contents of GPR(**rD**) are ANDed with UI || ¹⁶0.

For **se_andi**, the contents of GPR(**rX**) are ANDed with the value of UI5.

For **se_and[.]**, the contents of GPR(**rX**) are ANDed with the contents of GPR(**rY**).

For **se_andc**, the contents of GPR(**rX**) are ANDed with the one's complement of the contents of GPR(**rY**).

The result is placed into GPR(**rA**) or GPR(**rX**) (**se_and[ic][.]**)

Special Registers Altered: CR0 (if Rc = 1)

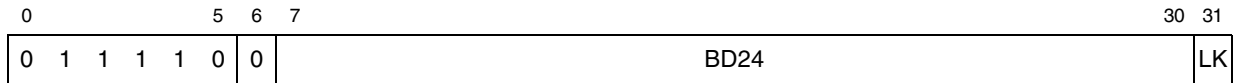
_bx

VLE	User
-----	------

_bx

Branch [and Link]

e_b BD24 (LK = 0)
e_bl BD24 (LK = 1)



$a \leftarrow CIA$
 $NIA \leftarrow (a + EXTS(BD24||0b0))_{32:63}$
 if LK=1 then LR $\leftarrow CIA + 4$

Let the BTEA be calculated as follows:

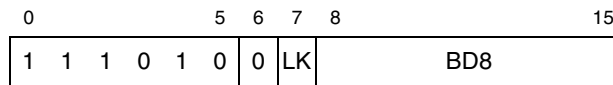
- For **e_b[l]**, let BTEA be the sum of the CIA and the sign-extended value of the BD24 instruction field concatenated with 0b0.

The BTEA is the address of the next instruction to be executed.

If LK = 1, the sum CIA+4 is placed into the LR.

Special Registers Altered: LR (if LK = 1)

se_b BD8 (LK = 0)
se_bl BD8 (LK = 1)



$a \leftarrow CIA$
 $NIA \leftarrow (a + EXTS(BD8||0b0))_{32:63}$
 if LK=1 then LR $\leftarrow CIA + 2$

Let the BTEA be calculated as follows:

- For **se_b[l]**, let BTEA be the sum of the CIA and the sign-extended value of the BD8 instruction field concatenated with 0b0.

The BTEA is the address of the next instruction to be executed.

If LK = 1, the sum CIA+2 is placed into the LR.

Special Registers Altered: LR (if LK = 1)

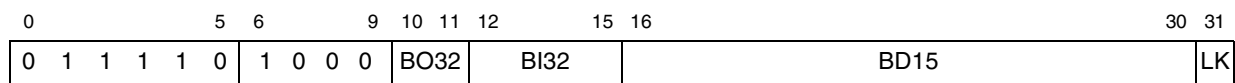
_bcx

VLE	User
-----	------

_bcx

Branch Conditional [and Link]

e_bc BO32,BI32,BD15 (LK = 0)
e_bcl BO32,BI32,BD15 (LK = 1)



```

if BO320 then CTR32:63 ← CTR32:63 - 1
ctr_ok ← ¬BO320 | ((CTR32:63 ≠ 0) ⊕ BO321)
cond_ok ← BO320 | (CRBI32+32 ≡ BO321)
if ctr_ok & cond_ok then
    NIA ← (CIA + EXTS(BD15 || 0b0))32:63
else
    NIA ← CIA + 4
if LK=1 then LR ← CIA + 4

```

Let the BTEA be calculated as follows:

- For **e_bc[l]**, let BTEA be the sum of the CIA and the sign-extended value of the BD15 instruction field concatenated with 0b0.

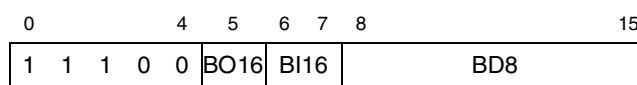
BO32 specifies any conditions that must be met for the branch to be taken, as defined in [Section 2.3.2, “Branch Instructions.”](#) The sum BI32+32 specifies the CR bit. Only CR[32–47] may be specified.

If the branch conditions are met, the BTEA is the address of the next instruction to be executed.

If LK = 1, the sum CIA + 4 is placed into the LR.

Special Registers Altered: CTR (if BO32₀ = 1)

LR (if LK = 1)

se_bc BO16,BI16,BD8

```

cond_ok ← (CRBI16+32 ≡ BO16)
if cond_ok then
    NIA ← (CIA + EXTS(BD8 || 0b0))32:63
else
    NIA ← CIA + 2

```

Let the BTEA be calculated as follows:

- For **se_bc**, BTEA is the sum of the CIA and the sign-extended value of the BD8 instruction field concatenated with 0b0.

BO16 specifies any conditions that must be met for the branch to be taken, as defined in [Section 2.3.2, “Branch Instructions.”](#) The sum BI16+32 specifies CR bit; only CR[32–35] may be specified.

If the branch conditions are met, the BTEA is the address of the next instruction to be executed.

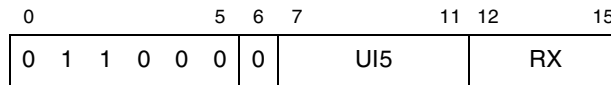
Special Registers Altered: None

_bclri

VLE	User
-----	------

_bclri

Bit Clear Immediate

se_bclri**rX,UI5**

```

a ← UI5
b ← a1 || 0 || 31-a1
result32:63 ← GPR(RX) & b
GPR(RX) ← result32:63

```

For **se_bclri**, the bit of GPR(**rX**) specified by the value of UI5 is cleared and all other bits in GPR(**rX**) remain unaffected.

Special Registers Altered: None

_bctrx

VLE	User
-----	------

_bctrx

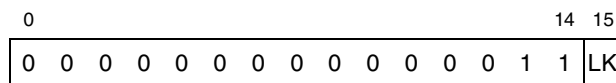
Branch to Count Register [and Link]

se_bctr

(LK = 0)

se_bctrl

(LK = 1)



$NIA \leftarrow CTR_{32:62} \parallel 0b0$
 if LK=1 then LR \leftarrow CIA + 2

Let the BTEA be calculated as follows:

- For **se_bctr[I]**, let BTEA be bits 32–62 of the contents of the CTR concatenated with 0b0.

The BTEA is the address of the next instruction to be executed.

If LK = 1, the sum CIA + 2 is placed into the LR.

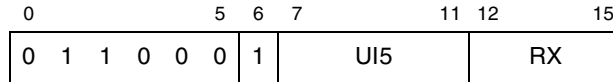
Special Registers Altered: LR (if LK = 1)

_bgeni

VLE	User
-----	------

_bgeni

Bit Generate Immediate

se_bgeni**rX,UI5**

$$a \leftarrow \text{UI5}$$

$$b \leftarrow {}^a0 \parallel 1 \parallel {}^{31-a}0$$

$$\text{GPR}(\text{RX}) \leftarrow b$$

For **se_bgeni**, a constant value consisting of a single '1' bit surrounded by '0's is generated and the value is placed into GPR(**rX**). The position of the '1' bit is specified by the UI5 field.

Special Registers Altered: None

_blrx

VLE	User
-----	------

_blrx

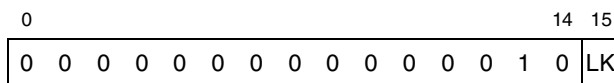
Branch to Link Register [and Link]

se_blr

(LK = 0)

se_blrl

(LK = 1)



$NIA \leftarrow LR_{32:62} \parallel 0b0$
 if LK=1 then $LR \leftarrow CIA + 2$

Let the BTEA be calculated as follows:

- For **se_blr**[I], let BTEA be bits 32–62 of the contents of the LR concatenated with 0b0.

The BTEA is the address of the next instruction to be executed.

If LK = 1, the sum CIA + 2 is placed into the LR.

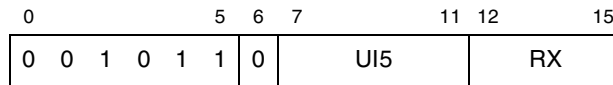
Special Registers Altered: LR (if LK = 1)

_bmaski

VLE	User
-----	------

_bmaski

Bit Mask Generate Immediate

se_bmaski**rX,UI5**

```

a ← UI5
if a = 0 then b ← 321 else b ← 32-a0 || a1
GPR(RX) ← b

```

For **se_bmaski**, a constant value consisting of a mask of low-order '1' bits that is zero-extended to 32 bits is generated, and the value is placed into GPR(**rX**). The number of low-order '1' bits is specified by the UI5 field. If UI5 is 0b000000, a value of all '1's is generated

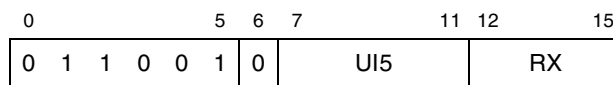
Special Registers Altered: None

_bseti

VLE	User
-----	------

_bseti

Bit Set Immediate

se_bseti**rX,UI5**

```

a ← UI5
b ← a0 || 1 || 31-a0
result32:63 ← GPR(RX) | b
GPR(RX) ← result32:63

```

For **se_bseti**, the bit of GPR(**rX**) specified by the value of UI5 is set, and all other bits in GPR(**rX**) remain unaffected.

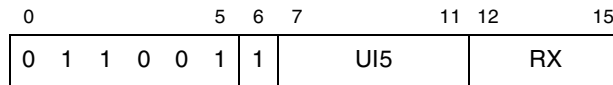
Special Registers Altered: None

_btsti

VLE	User
-----	------

_btsti

Bit Test Immediate

se_btsti**rX,UI5**

```

a ← UI5
b ← a0 || 1 || 31-a0
c ← GPR(RX) & b
if c = 320 then d ← 0b001 else d ← 0b010
CR0:3 ← d || XER50

```

For **se_btsti**, the bit of GPR(**rX**) specified by the value of UI5 is tested for equality to '1'. The result of the test is recorded in the CR. EQ is set if the tested bit is clear, LT is cleared, and GT is set to the inverse value of EQ.

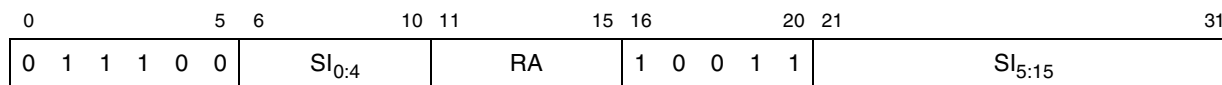
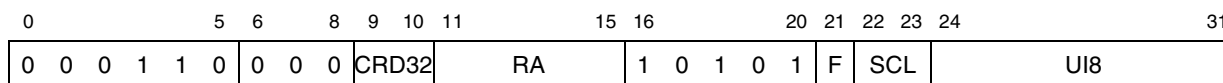
Special Registers Altered: CR[0–3]

_cmp

VLE	User
-----	------

_cmp

Compare [Immediate]

e_cmp16i **rA,SI****e_cmpi** **crD32,rA,SCI8**

```

a ← GPR(RA)32:63
if 'e_cmpi' then b ← SCI8(F,SCL,UI8)
if 'e_cmp16i' then b ← EXTS(SI0:4 || SI5:15)
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
if 'e_cmpi' then CR4×CRD32+32:4×CRD32+35 ← c || XERS0 // only CR0–CR3
if 'e_cmp16i' then CR32:35 ← c || XERS0 // only CR0

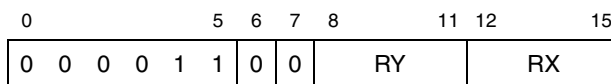
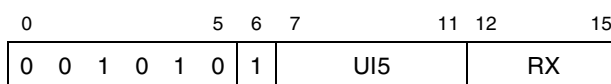
```

If **e_cmpi**, GPR(**rA**) contents are compared with the value of **SCI8**, treating operands as signed integers.

If **e_cmp16i**, GPR(**rA**) contents are compared with the sign-extended value of the **SI** field, treating operands as signed integers.

The result of the comparison is placed into **CR** field **crD** (**crD32**). For **e_cmpi**, only **CR0–CR3** may be specified. For **e_cmp16i**, only **CR0** may be specified.

Special Registers Altered: **CR** field **crD** (**crD32**) (**CR0** for **e_cmp16i**)

se_cmp **rX,rY****se_cmpi** **rX,UI5**

```

a ← GPR(RX)32:63
if 'se_cmpi' then b ← 270 || UI5
if 'se_cmp' then b ← GPR(RY)32:63
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001

```

$$CR_{0:3} \leftarrow c \parallel XER_{SO}$$

If **se_cmp**, the contents of GPR(**rX**) are compared with the contents of GPR(**rY**), treating the operands as signed integers. The result of the comparison is placed into CR field 0.

If **se_cmpi**, the contents of GPR(**rX**) are compared with the value of the zero-extended UI5 field, treating the operands as signed integers. The result of the comparison is placed into CR field 0.

Special Registers Altered: CR[0–3]

_cmph

VLE	User
-----	------

_cmph

Compare Halfword [Immediate]

e_cmph **crD,rA,rB**

0	5	6	10	11	15	16	20	21	30	31											
0	1	1	1	1	1	1	CRD	/	RA	RB	0	0	0	0	0	0	1	1	1	0	/

```

a ← EXTS(GPR(RA)48:63)
b ← EXTS(GPR(RB)48:63)
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR4×CRD+32:4×CRD+35 ← c || XERSO

```

For **e_cmph**, the contents of the low-order 16 bits of GPR(**rA**) and GPR(**rB**) are compared, treating the operands as signed integers. The result of the comparison is placed into CR field CRD.

Special Registers Altered: CR field CRD

se_cmph **rX,rY**

0	5	6	7	8	11	12	15		
0	0	0	0	1	1	1	0	RY	RX

```

a ← EXTS(GPR(RX)48:63)
b ← EXTS(GPR(RY)48:63)
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR0:3 ← c || XERSO

```

For **se_cmph**, the contents of the low-order 16 bits of GPR(**rX**) and GPR(**rY**) are compared, treating the operands as signed integers. The result of the comparison is placed into CR field 0.

Special Registers Altered: CR[0–3]

e_cmph16i **rA,SI**

0	5	6	10	11	15	16	20	21	31				
0	1	1	1	0	0	SI _{0:4}	RA	1	0	1	1	0	SI _{5:15}

```

a ← EXTS(GPR(RA)48:63)
b ← EXTS(SI0:4 || SI5:15)
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR32:35 ← c || XERSO // only CR0

```

VLE Instruction Set

The contents of the lower 16-bits of GPR(**rA**) are sign-extended and compared with the sign-extended value of the SI field, treating the operands as signed integers.

The result of the comparison is placed into CR0.

Special Registers Altered: CR0

cmphl

VLE	User
-----	------

cmphl _cmphl_

Compare Halfword Logical [Immediate]

e_cmphl **crD,rA,rB**

0	5	6	10	11	15	16	20	21	30	31											
0	1	1	1	1	1	1	CRD	/	RA	RB	0	0	0	0	1	0	1	1	1	0	/

```

a ← EXTZ(GPR(RA))48:63
b ← EXTZ(GPR(RB))48:63
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR4×CRD+32:4×CRD+35 ← c || XERSO

```

For **e_cmphl**, the contents of the low-order 16 bits of GPR(**rA**) and GPR(**rB**) are compared, treating the operands as unsigned integers. The result of the comparison is placed into CR field CRD.

Special Registers Altered: CR field CRD

se_cmphl **rX,rY**

0	5	6	7	8	11	12	15		
0	0	0	0	1	1	1	1	RY	RX

```

a ← GPR(RX)48:63
b ← GPR(RY)48:63
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR0:3 ← c || XERSO

```

For **se_cmphl**, the contents of the low-order 16 bits of GPR(**rX**) and GPR(**rY**) are compared, treating the operands as unsigned integers. The result of the comparison is placed into CR field 0.

Special Registers Altered: CR[0–3]

e_cmphl16i **rA,UI**

0	5	6	10	11	15	16	20	21	31					
0	1	1	1	0	0	0	UI _{0:4}	RA	1	0	1	1	1	UI _{5:15}

```

a ← 160 || GPR(RA)48:63
b ← 160 || UI0:4 || UI5:15
if a < b then c ← 0b100
if a > b then c ← 0b010
if a = b then c ← 0b001
CR32:35 ← c || XERSO // only CR0

```

VLE Instruction Set

The contents of the lower 16-bits of GPR(**rA**) are zero-extended and compared with the zero-extended value of the UI field, treating the operands as unsigned integers.

The result of the comparison is placed into CR0.

Special Registers Altered: CR0

cmpl

VLE	User
-----	------

cmpl _cmpl_

Compare Logical [Immediate]

e_cmpl16i **rA,UI**

0	5	6	10	11	15	16	20	21	31									
0	1	1	1	0	0	UI _{0:4}		RA		1	0	1	0	1	UI _{5:15}			

e_cmpli **crD32,rA,SCI8**

0	5	6	8	9	10	11	15	16	20	21	22	23	24	31							
0	0	0	1	1	0	0	0	1	CRD32		RA		1	0	1	0	1	F	SCL	UI8	

```

a ← GPR(RA)32:63
if 'e_cmpli' then b ← SCI8(F,SCL,UI8)
if 'e_cmpl16i' then b ← 160 || UI0:4 || UI5:15
if a <u b then c ← 0b100
if a >u b then c ← 0b010
if a = b then c ← 0b001
if 'e_cmpli' then CR4×CRD32+32:4×CRD32+35 ← c || XERSO // only CR0–CR3
if 'e_cmpl16i' then CR32:35 ← c || XERSO // only CR0

```

If **e_cmpli**, the contents of bits 32–63 of GPR(**rA**) are compared with the SCI8 value, treating operands as unsigned integers.

If **e_cmpl16i**, GPR(**rA**) and the zero-extended UI value contents are compared, treating operands as unsigned integers.

The result of the comparison is placed into CR field CRD (CRD32). For **e_cmpli**, only CR0–CR3 may be specified. For **e_cmpl16i**, only CR0 may be specified.

Special Registers Altered: CR field CRD (CRD32) (CR0 for **e_cmpl16i**)

se_cmpl **rX,rY**

0	5	6	7	8	11	12	15		
0	0	0	0	1	1	0	1	RY	RX

se_cmpli **rX,OIMM**

0	5	6	7	11	12	15		
0	0	1	0	0	0	1	OIM5 ¹	RX

¹ OIMM = OIM5 + 1

```

a ← GPR(RX)32:63
if 'se_cmpli' then b ← 270 || OFFSET(OIM5)
if 'se_cmpl' then b ← GPR(RY)32:63

```

VLE Instruction Set

```
if a <u b then c ← 0b100
if a >u b then c ← 0b010
if a = b then c ← 0b001
CR0:3 ← c || XERS0
```

If **se_cmpl**, the contents of GPR(**rX**) and GPR(**rY**) are compared, treating operands as unsigned integers. The result is placed into CR field 0.

If **se_cmpli**, the contents of GPR(**rX**) are compared with the zero-extended offset value of OIM5 (a final value in the range 1–32), treating the operands as unsigned integers. The result is placed into CR field 0.

Special Registers Altered: CR[0–3]

crand

VLE	User
-----	------

crand _crand_

Condition Register AND

e_crاند **crbD,crbA,crbB**

0	5	6	10	11	15	16	20	21	30	31											
0	1	1	1	1	1	1	CRBD	CRBA	CRBB	0	1	0	0	0	0	0	0	0	0	1	/

$$CR_{BT+32} \leftarrow CR_{BA+32} \& CR_{BB+32}$$

The content of bit CRBA+32 of the CR is ANDed with the content of bit CRBB+32 of the CR, and the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

Condition Register AND with Complement

e_crاندc **crbD,crbA,crbB**

0	5	6	10	11	15	16	20	21	30	31												
0	1	1	1	1	1	1	CRBD	CRBA	CRBB	0	0	1	0	0	0	0	0	0	0	0	1	/

$$CR_{BT+32} \leftarrow CR_{BA+32} \& \neg CR_{BB+32}$$

The content of bit CRBA+32 of the CR is ANDed with the one's complement of the content of bit CRBB+32 of the CR, and the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

CR Equivalent

e_crandv **crbD,crbA,crbB**

0	5	6	10	11	15	16	20	21	30	31												
0	1	1	1	1	1	1	CRBD	CRBA	CRBB	0	1	0	0	1	0	0	0	0	0	0	1	/

$$CR_{BT+32} \leftarrow CR_{BA+32} \oplus CR_{BB+32}$$

The content of bit CRBA+32 of the CR is XORed with the content of bit CRBB+32 of the CR, and the one's complement of result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

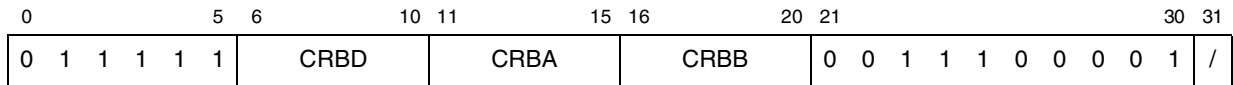
_crnand

VLE	User
-----	------

__crnand __crnand __

Condition Register NAND

e_crnand crbD,crbA,crbB



$$CR_{BT+32} \leftarrow \neg(CR_{BA+32} \& CR_{BB+32})$$

The content of bit CRBA+32 of the CR is ANDed with the content of bit CRBB+32 of the CR, and the one's complement of the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

_crnor

VLE	User
-----	------

_crnor

Condition Register NOR

e_crnor **crbD,crbA,crbB**

0	5	6	10	11	15	16	20	21	30	31									
0	1	1	1	1	1	CRBD	CRBA	CRBB	0	0	0	0	1	0	0	0	0	1	/

$$CR_{BT+32} \leftarrow \neg(CR_{BA+32} \mid CR_{BB+32})$$

The content of bit CRBA+32 of the CR is ORed with the content of bit CRBB+32 of the CR, and the one's complement of the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

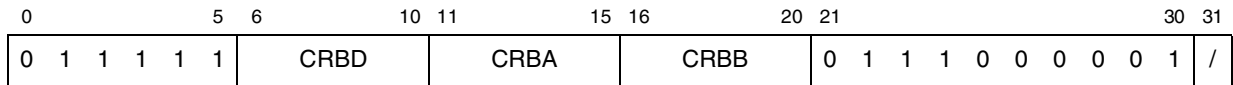
_cror

VLE	User
-----	------

__cror

Condition Register OR

e_cror **crbD,crbA,crbB**



$$CR_{BT+32} \leftarrow CR_{BA+32} \mid CR_{BB+32}$$

The content of bit CRBA+32 of the CR is ORed with the content of bit CRBB+32 of the CR, and the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

_cror

VLE	User
-----	------

___cror ___cror ___

Condition Register OR with Complement

e_crorc **crbD,crbA,crbB**

0	5	6	10	11	15	16	20	21	30	31									
0	1	1	1	1	1	CRBD	CRBA	CRBB	0	1	1	0	1	0	0	0	0	1	/

$$CR_{BT+32} \leftarrow CR_{BA+32} \mid \neg CR_{BB+32}$$

The content of bit CRBA+32 of the CR is ORed with the one's complement of the content of bit CRBB+32 of the CR, and the result is placed into bit CRBD+32 of the CR.

Special Registers Altered: CR

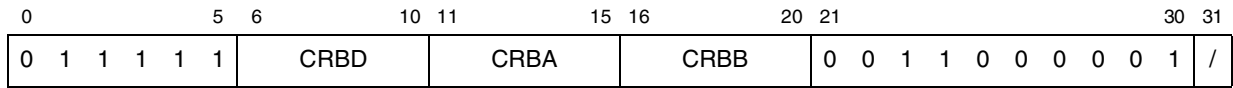
_crxor

VLE	User
-----	------

__crxor __crxor __

Condition Register XOR

e_crxor crbD,crbA,crbB



$$CR_{crbD+32} \leftarrow CR_{crbA+32} \oplus CR_{crbB+32}$$

The content of bit CRBA+32 of the CR is XORed with the content of bit CRBB+32 of the CR, and the result is placed into bit CRBD+32 of the CR.

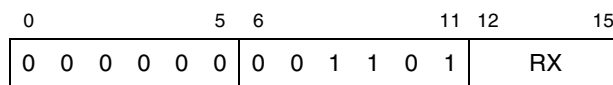
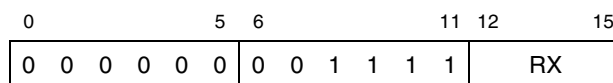
Special Registers Altered: CR

_extsbx

VLE	User
-----	------

_extsbx

Extend Sign (Byte | Halfword)

se_extsb**rX****se_extsh****rX**

```

if se_extsb then n ← 56
if se_extsh then n ← 48
if `extsw`      then n ← 32
if Rc=1 then do
    LT ← GPR(RS)n:63 < 0
    GT ← GPR(RS)n:63 > 0
    EQ ← GPR(RS)n:63 = 0
    CR0 ← LT || GT || EQ || SO
s ← GPR(RS or RX)n
GPR(RA or RX) ← n-32s || GPR(RS or RX)n:63

```

For **se_extsb**, the contents of bits 56–63 of GPR(**rX**) are placed into bits 56–63 of GPR(**rX**). Bit 56 of the contents of GPR(**rX**) is copied into bits 32–55 of GPR(**rX**).

For **se_extsh**, the contents of bits 48–63 of GPR(**rX**) are placed into bits 48–63 of GPR(**rX**). Bit 48 of the contents of GPR(**rX**) is copied into bits 32–47 of GPR(**rX**).

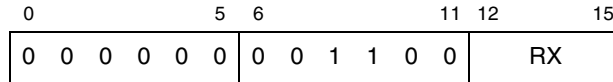
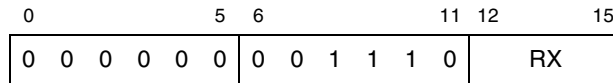
Special Registers Altered: CR0 (if Rc=1)

_extzx

VLE	User
-----	------

extzx

Extend Zero (Byte | Halfword)

se_extzb**rX****se_extzh****rX**

```

if 'se_extzb' then n ← 56
if 'se_extzh' then n ← 48
GPR(RX) ← n-320 || GPR(RX)n:63

```

For **se_extzb**, the contents of bits 56–63 of GPR(**rX**) are placed into bits 56–63 of GPR(**rX**). Bits 32–55 of GPR(**rX**) are cleared.

For **se_extzh**, the contents of bits 48–63 of GPR(**rX**) are placed into bits 48–63 of GPR(**rX**). Bits 32–47 of GPR(**rX**) are cleared.

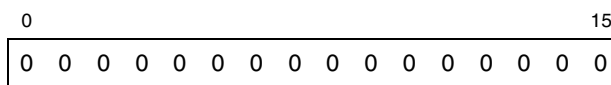
Special Registers Altered: None

_illegal

VLE	User
-----	------

_illegal

Illegal

se_illegal

```

SRR1 ← MSR
SRR0 ← CIA
NIA ← IVPR32:47 || IVOR648:59 || 0b0000
MSRWE,EE,PR,IS,DS,FP,FE0,FE1 ← 0b0000_0000

```

se_illegal is used to request an illegal instruction exception. A program interrupt is generated. The contents of the MSR are copied into SRR1 and the address of the **se_illegal** instruction is placed into SRR0.

MSR[WE,EE,PR,IS,DS,FP,FE0,FE1] are cleared.

The interrupt causes the next instruction to be fetched from address IVPR[32–47]||IVOR6[48–59]||0b0000

This instruction is context synchronizing.

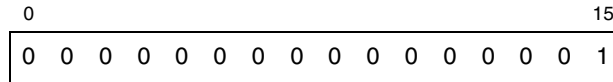
Special Registers Altered: SRR0 SRR1 MSR[WE,EE,PR,IS,DS,FP,FE0,FE1]

_isync

VLE	User
-----	------

_isync

Instruction Synchronize

se_isync

The **se_isync** instruction provides an ordering function for the effects of all instructions executed by the processor executing the **se_isync** instruction. Executing an **se_isync** instruction ensures that all instructions preceding the **se_isync** instruction have completed before the **se_isync** instruction completes, and that no subsequent instructions are initiated until after the **se_isync** instruction completes. It also causes any prefetched instructions to be discarded, with the effect that subsequent instructions are fetched and executed in the context established by the instructions preceding the **se_isync** instruction.

The **se_isync** instruction may complete before memory accesses associated with instructions preceding the **se_isync** instruction have been performed.

This instruction is context synchronizing (see Book E). It has identical semantics to Book E **isync**, just a different encoding.

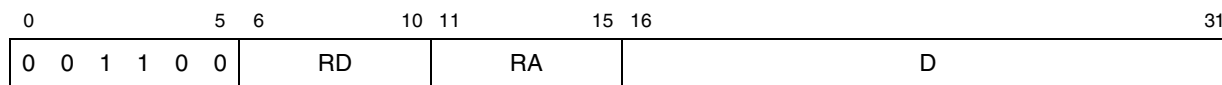
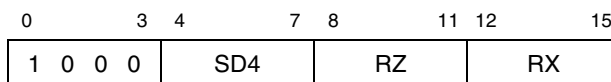
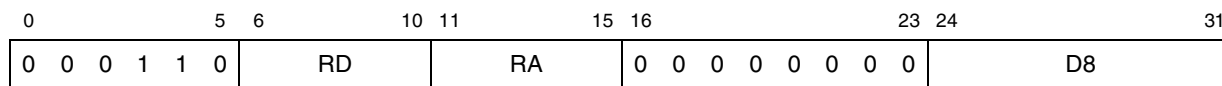
Special Registers Altered: None

_lbzx

VLE	User
-----	------

_lbzx

Load Byte and Zero [with Update] [Indexed]

e_lbz **rD,D(rA)** (D-mode)**se_lbz** **rZ,SD4(rX)** (SD4-mode)**e_lbzu** **rD,D8(rA)** (D8-mode)

```

if (RA=0 & !se_lbz) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (280 || SD4))32:63
GPR(RD or RZ) ← 240 || MEM(EA,1)
if e_lbzu then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_lbz** and **e_lbzu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_lbz**, let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field.

The byte in memory addressed by EA is loaded into bits 56–63 of GPR(**rD** or **rZ**). Bits 32–55 of GPR(**rD** or **rZ**) are cleared.

If **e_lbzu**, EA is placed into GPR(**rA**).If **e_lbzu** and **rA** = 0 or **rA** = **rD**, the instruction form is invalid.

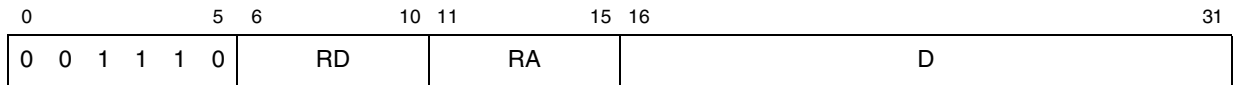
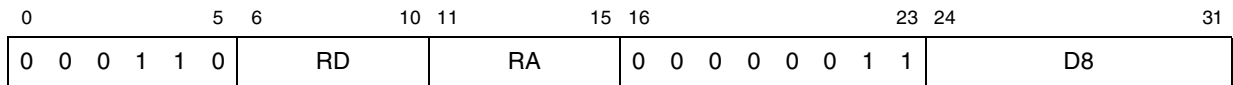
Special Registers Altered: None

_lhax

VLE	User
-----	------

_lhax

Load Halfword Algebraic [with Update] [Indexed]

e_lha **rD,D(rA)** (D-mode)**e_lhau** **rD,D8(rA)** (D8-mode)

```

if RA=0 then a ← 320 else a ← GPR(RA)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
GPR(RD) ← EXTS(MEM(EA,2))32:63
if e_lhau then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_lha** and **e_lhau**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.

The half word in memory addressed by EA is loaded into bits 48–63 of GPR(**rD**). Bits 32–47 of GPR(**rD**) are filled with a copy of bit 0 of the loaded half word.

If **e_lhau**, EA is placed into GPR(**rA**).

If **e_lhau** and **rA** = 0 or **rA** = **rD**, the instruction form is invalid.

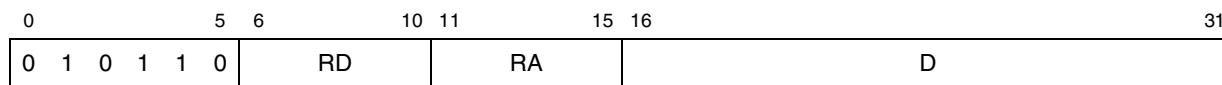
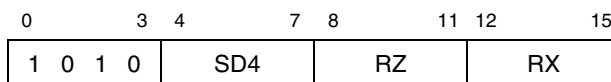
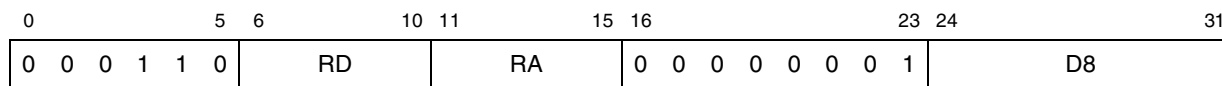
Special Registers Altered: None

_lhzx

VLE	User
-----	------

_lhzx

Load Halfword and Zero [with Update] [Indexed]

e_lhz **rD,D(rA)** (D-mode)**se_lhz** **rZ,SD4(rX)** (SD4-mode)**e_lhzu** **rD,D8(rA)** (D8-mode)

```

if (RA=0 & !se_lhz) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (270 || SD4 || 0))32:63
GPR(RD or RZ) ← 160 || MEM(EA,2)
if e_lhzu then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_lhz** and **e_lhzu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_lhz** let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field shifted left by 1 bit.

The half word in memory addressed by EA is loaded into bits 48–63 of GPR(**rD**). Bits 32–47 of GPR(**rD**) are cleared.

If **e_lhzu**, EA is placed into GPR(**rA**).If **e_lhzu** and **rA** = 0 or **rA** = **rD**, the instruction form is invalid.

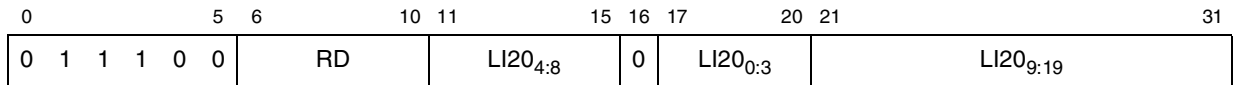
Special Registers Altered: None

_lix

VLE	User
-----	------

_lix

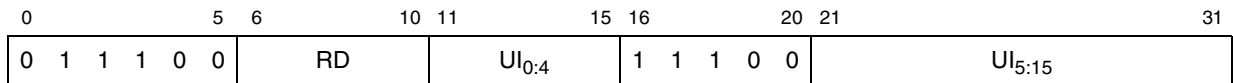
Load Immediate [Shifted]

e_li **rD,LI20** (LI20-mode)

$$\begin{aligned} \text{LI20} &\leftarrow \text{LI20}_{0:3} \parallel \text{LI20}_{4:8} \parallel \text{LI20}_{9:19} \\ \text{GPR}(\text{RD}) &\leftarrow \text{EXTS}(\text{LI20}) \end{aligned}$$

For **e_li**, the sign-extended LI20 field is placed into GPR(**rD**).

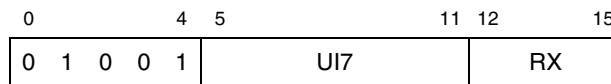
Special Registers Altered: None

e_lis **rD,UI**

$$\begin{aligned} \text{UI} &\leftarrow \text{UI}_{0:4} \parallel \text{UI}_{5:15} \\ \text{GPR}(\text{RD}) &\leftarrow \text{UI} \parallel \text{16}0 \end{aligned}$$

For **e_lis**, the UI field is concatenated on the right with 16 0's and placed into GPR(**rD**).

Special Registers Altered: None

se_li **rX,UI7**

$$\text{GPR}(\text{RX}) \leftarrow \text{25}0 \parallel \text{UI7}$$

For **se_li**, the zero-extended UI7 field is placed into GPR(**rX**).

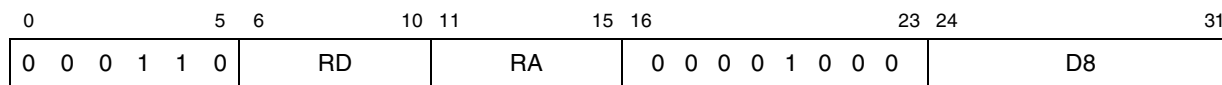
Special Registers Altered: None

_Imw

VLE	User
-----	------

_Imw

Load Multiple Word

e_Imw **rD,D8(rA)**

```

if RA=0 then EA ← EXTS(D8)32:63
else      EA ← (GPR(RA)+EXTS(D8))32:63
r ← RD
do while r ≤ 31
    GPR(r) ← MEM(EA, 4)
    r ← r + 1
    EA ← (EA+4)32:63

```

Let the EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D8 instruction field.

Let $n = (32 - rD)$. n consecutive words starting at EA are loaded into bits 32–63 of registers GPR(**rD**) through GPR(31).

EA must be a multiple of 4. If it is not, either an alignment interrupt is invoked or the results are boundedly undefined. If **rA** is in the range of registers to be loaded, including the case in which **rA** = 0, the instruction form is invalid.

Special Registers Altered: None

_lwz

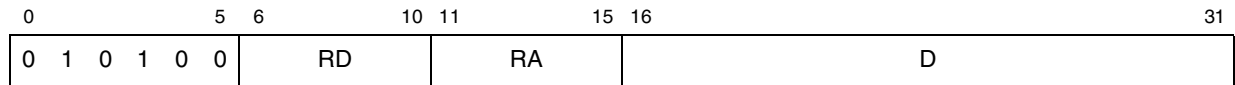
VLE	User
-----	------

_lwz

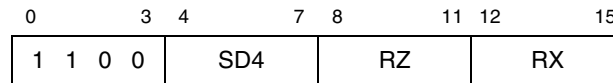
Load Word and Zero [with Update] [Indexed]

e_lwz**rD,D(rA)**

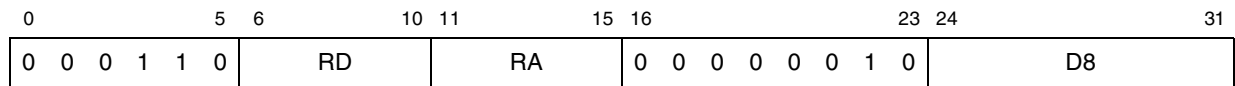
(D-mode)

**se_lwz****rZ,SD4(rX)**

(SD4-mode)

**e_lwzu****rD,D8(rA)**

(D8-mode)



```

if (RA=0 & !se_lwz) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (260 || SD4 || 20))32:63
GPR(RD or RZ) ← MEM(EA,4)
if e_lwzu then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_lwz** and **e_lwzu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_lwz** let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field shifted left by 2 bits.

The word in memory addressed by the EA is loaded into bits 32–63 of GPR(**rD**).If **e_lwzu**, EA is placed into GPR(**rA**).If **e_lwzu** and **rA** = 0 or **rA** = **rD**, the instruction form is invalid.

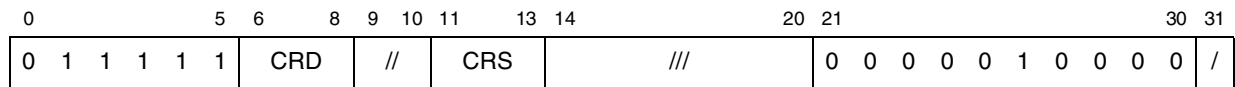
Special Registers Altered: None

_mcrf

VLE	User
-----	------

_mcrf

Move CR Field

e_mcrf**crD,crS**

$$CR_{4 \times CRD + 32 : 4 \times CRD + 35} \leftarrow CR_{4 \times CRS + 32 : 4 \times CRS + 35}$$

The contents of field **crS** (bits $4 \times CRS + 32$ through $4 \times CRS + 35$) of the CR are copied to field **crD** (bits $4 \times CRD + 32$ through $4 \times CRD + 35$) of the CR.

Special Registers Altered: CR

_mfar

VLE	User
-----	------

_mfar

Move from Alternate Register

se_mfar**rX,arY**

0	5	6	7	8	11	12	15
0	0	0	0	0	1	1	ARY
						RX	

$$\text{GPR}(\text{RX}) \leftarrow \text{GPR}(\text{ARY})$$

For **se_mfar**, the contents of GPR(**arY**) are placed into GPR(**rX**). **arY** specifies a GPR in the range R8–R23. The encoding 0000 specifies R8, 0001 specifies R9, ..., 1111 specifies R23.

Special Registers Altered: None

_mfctr

VLE	User
-----	------

_mfctr

Move From Count Register

se_mfctr**rX**

0	5	6	11	12	15
0	0	0	0	0	0
0	0	0	1	0	1
					RX

GPR(RX) ← CTR

The CTR contents are placed into bits 32–63 of GPR(rX).

Special Registers Altered: None

_mflr

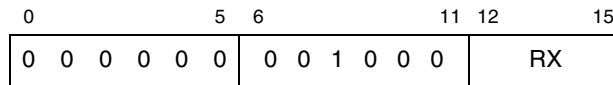
VLE	User
-----	------

_mflr

Move From Link Register

se_mflr

rX



$$\text{GPR}(\text{rX}) \leftarrow \text{LR}$$

The LR contents are placed into bits 32–63 of GPR(rX).

Special Registers Altered: None

_mr

VLE	User
-----	------

_mr

Move Register

se_mr**rX,rY**

0	5	6	7	8	11	12	15
0	0	0	0	0	0	1	RY
						RX	

$$\text{GPR}(\text{RX}) \leftarrow \text{GPR}(\text{RY})$$

For **se_mr**, the contents of GPR(**rY**) are placed into GPR(**rX**).

Special Registers Altered: None

_mtar

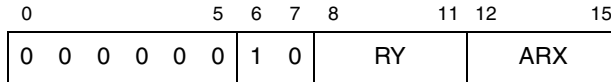
VLE	User
-----	------

_mtar

Move to Alternate Register

se_mtar

arX,rY



$$GPR(ARX) \leftarrow GPR(RY)$$

For **se_mtar**, the contents of GPR(**rY**) are placed into GPR(**arX**). **arX** specifies a GPR in the range R8–R23. The encoding 0000 specifies R8, 0001 specifies R9,..., 1111 specifies R23.

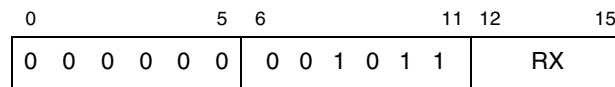
Special Registers Altered: None

_mtctr

VLE	User
-----	------

_mtctr

Move To Count Register

se_mtctr**rX**

CTR ← GPR(RX)

The contents of bits 32–63 of GPR(**rX**) are placed into the CTR.

Special Registers Altered: CTR

_mtr

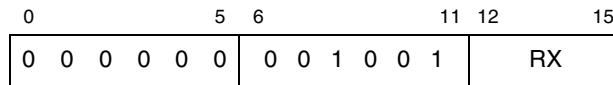
VLE	User
-----	------

_mtr

Move To Link Register

se_mtr

rX



$$LR \leftarrow GPR(RX)$$

The contents of bits 32–63 of GPR(**rX**) are placed into the LR.

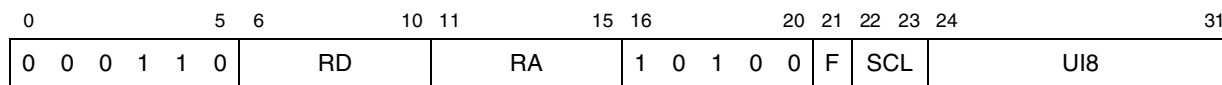
Special Registers Altered: LR

_mullix

VLE	User
-----	------

_mullix

Multiply Low [2 operand] Immediate

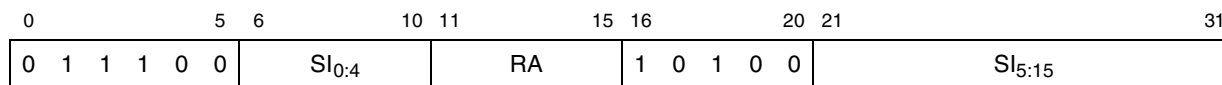
e_mulli **rD,rA,SCI8**

$$\begin{aligned} \text{imm} &\leftarrow \text{SCI8}(\text{F}, \text{SCL}, \text{UI8}) \\ \text{prod}_{0:63} &\leftarrow \text{GPR}(\text{rA}) \times \text{imm} \\ \text{GPR}(\text{rD}) &\leftarrow \text{prod}_{32:63} \end{aligned}$$

Bits 32–63 of the 64-bit product of the contents of GPR(**rA**) and the value of SCI8 are placed into GPR(**rD**).

Both operands and the product are interpreted as signed integers.

Special Registers Altered: None

e_mull2i **rA,SI**

$$\begin{aligned} \text{prod}_{0:63} &\leftarrow \text{GPR}(\text{rA}) \times \text{EXTS}(\text{SI}_{0:4} \parallel \text{SI}_{5:15}) \\ \text{GPR}(\text{rA}) &\leftarrow \text{prod}_{32:63} \end{aligned}$$

Bits 32–63 of the 64-bit product of the contents of GPR(**rA**) and the sign-extended value of the SI field are placed into GPR(**rA**).

Both operands and the product are interpreted as signed integers.

Special Registers Altered: None

_mullwx

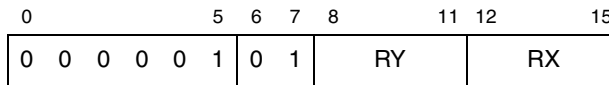
VLE	User
-----	------

_mullwx

Multiply Low Word

se_mullw

rX,rY



$$\text{prod}_{0:63} \leftarrow \text{GPR}(\text{RX})_{32:63} \times \text{GPR}(\text{RY})_{32:63}$$

$$\text{GPR}(\text{RX}) \leftarrow \text{prod}_{32:63}$$

Bits 32–63 of the 64-bit product of the contents of bits 32–63 of GPR(**rX**) and the contents of bits 32–63 of GPR(**rY**) is placed into GPR(**rX**).

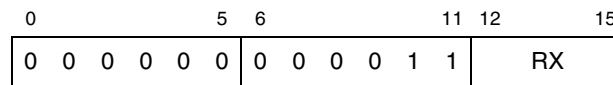
Special Registers Altered: None

_negx

VLE	User
-----	------

_negx

Negate

se_neg**rX**

```

result32:63 ← -GPR(RX) + 1
GPR(RX) ← result32:63

```

The sum of the one's complement of the contents of GPR(**rX**) and 1 is placed into GPR(**rX**).

If bits 32–63 of GPR(**rX**) contain the most negative 32-bit number (0x8000_0000), bits 32–63 of the result contain the most negative 32-bit number

Special Registers Altered: None

_notx

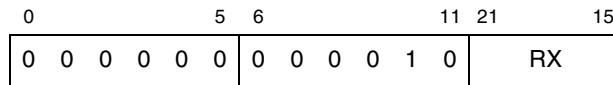
VLE	User
-----	------

_notx

NOT

se_not

rX



$result_{32:63} \leftarrow \neg GPR(RX)$
 $GPR(RX) \leftarrow result_{32:63}$

The contents of GPR(**rX**) are inverted.

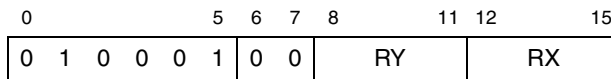
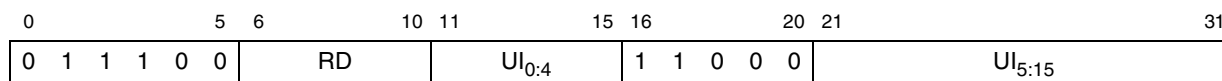
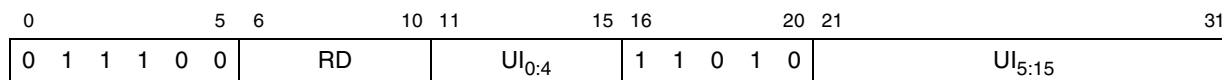
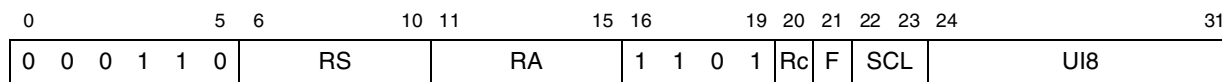
Special Registers Altered: None

_orX

VLE	User
-----	------

_orX

OR [2 operand] [Immediate | with Complement] [Shifted][and Record]

se_or **rX,rY****e_or2i** **rD,UI****e_or2is** **rD,UI****e_ori** **rA,rS,SCI8** (Rc = 0)**e_ori.** **rA,rS,SCI8** (Rc = 1)

```

if `e_ori[.]` then b ← SCI8(F,SCL,UI8)
if `e_or2i` then b ← 160 || UI0:4 || UI5:15
if `e_or2is` then b ← UI0:4 || UI5:15 || 160
if `se_or` then b ← GPR(RB)
result0:63 ← GPR(RS or RD or RX) | b
if Rc=1 then do
    LT ← result32:63 < 0
    GT ← result32:63 > 0
    EQ ← result32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RA or RD or RX) ← result

```

For **e_ori[.]**, the contents of GPR(**rS**) are ORed with the value of SCI8.For **e_or2i**, the contents of GPR(**rD**) are ORed with ¹⁶0 || UI.For **e_or2is**, the contents of GPR(**rD**) are ORed with UI || ¹⁶0.For **se_or**, the contents of GPR(**rX**) are ORed with the contents of GPR(**rY**).The result is placed into GPR(**rA** or **rX**).

The preferred ‘no-op’ (an instruction that does nothing) is:

```
e_ori 0,0,0
```

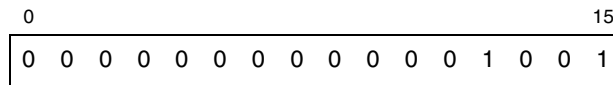
Special Registers Altered: CR0 (if Rc = 1)

_rfci

VLE	Supervisor
-----	------------

_rfci

Return From Critical Interrupt

se_rfci

MSR ← CSRR1
 NIA ← CSRR0_{0:62} || 0b0

The **se_rfci** instruction is used to return from a critical class interrupt, or as a means of establishing a new context and synchronizing on that new context simultaneously.

The contents of CSRR1 are placed into the MSR. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address CSRR0[32–62]||0b0. If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0 or CSRR0 by the interrupt processing mechanism (see Book E) is the address of the instruction that would have been executed next had the interrupt not occurred (that is, the address in CSRR0 at the time of the execution of the **se_rfci**).

Execution of this instruction is privileged and restricted to supervisor mode.

Execution of this instruction is context synchronizing.

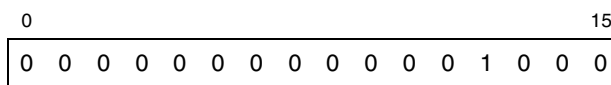
Special Registers Altered: MSR

_rfi

VLE | Supervisor

_rfi

Return From Interrupt

se_rfi

MSR ← SRR1
 NIA ← SRR0_{0:62} || 0b0

The **se_rfi** instruction is used to return from a non-critical class interrupt, or as a means of simultaneously establishing a new context and synchronizing on that new context.

The contents of SRR1 are placed into the MSR. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched under control of the new MSR value from the address SRR0[32–62]||0b0. If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0 or CSRR0 by the interrupt processing mechanism (see Book E) is the address of the instruction that would have been executed next had the interrupt not occurred (that is, the address in SRR0 at the time of the execution of the **se_rfi**).

Execution of this instruction is privileged and restricted to supervisor mode.

Execution of this instruction is context synchronizing.

Special Registers Altered: MSR

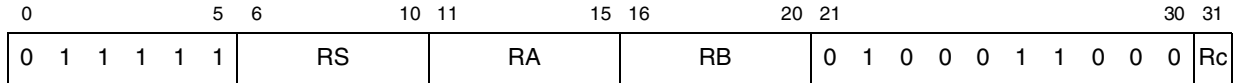
_rlw

VLE	User
-----	------

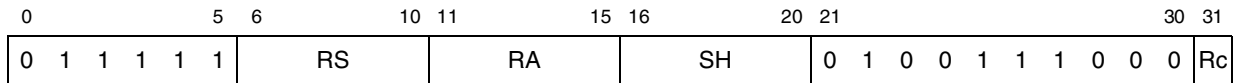
_rlw

Rotate Left Word [Immediate]

e_rlw **rA,rS,rB** (Rc = 0)
e_rlw. **rA,rS,rB** (Rc = 1)



e_rlwi **rA,rS,SH** (Rc = 0)
e_rlwi. **rA,rS,SH** (Rc = 1)



```

if `e_rlw[.]` then n ← GPR(RB)59:63
else
    n ← SH
result32:63 ← ROTL32(GPR(RS)32:63,n)
if Rc=1 then do
    LT ← result32:63 < 0
    GT ← result32:63 > 0
    EQ ← result32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RA) ← result32:63
    
```

If **e_rlw[.]**, let the shift count *n* be the contents of bits 59–63 of GPR(**rB**).

If **e_rlwi[.]**, let the shift count *n* be SH.

The contents of GPR(**rS**) are rotated₃₂ left *n* bits. The rotated data is placed into GPR(**rA**).

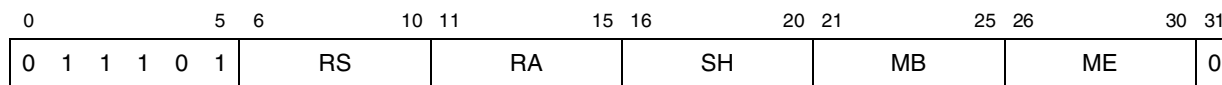
Special Registers Altered: CR0 (if Rc = 1)

_rlwimi

VLE	User
-----	------

_rlwimi

Rotate Left Word Immediate then Mask Insert

e_rlwimi **rA,rS,SH,MB,ME**

```

n ← SH
b ← MB+32
e ← ME+32
r ← ROTL32(GPR(rS)32:63, n)
m ← MASK(b, e)
result32:63 ← r&m | GPR(rA)&¬m
GPR(rA) ← result32:63

```

Let the shift count n be the value SH.

The contents of GPR(**rS**) are rotated₃₂ left n bits. A mask is generated having 1 bit from bit MB+32 through bit ME+32 and 0 bits elsewhere. The rotated data are inserted into GPR(**rA**) under control of the generated mask (if a mask bit is 1, the associated bit of the rotated data is placed into the target register, and if the mask bit is 0, the associated bit in the target register remains unchanged).

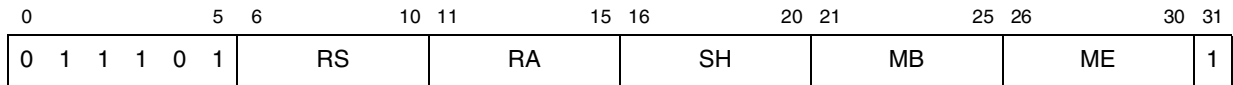
Special Registers Altered: None

_rlwinm

VLE	User
-----	------

_rlwinm

Rotate Left Word Immediate then AND with Mask

e_rlwinm **rA,rS,SH,MB,ME**

```

n ← SH
b ← MB+32
e ← ME+32
r ← ROTL32(GPR(rS)32:63, n)
m ← MASK(b, e)
result32:63 ← r & m
GPR(rA) ← result32:63

```

Let the shift count n be SH.

The contents of GPR(**rS**) are rotated₃₂ left n bits. A mask is generated having 1 bit from bit MB+32 through bit ME+32 and 0 bits elsewhere. The rotated data are ANDed with the generated mask and the result is placed into GPR(**rA**).

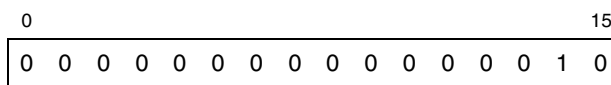
Special Registers Altered: None

_SC

VLE	User
-----	------

_SC

System Call

se_sc

$SRR1 \leftarrow MSR$
 $SRR0 \leftarrow CIA+2$
 $NIA \leftarrow IVPR_{32:47} \parallel IVOR8_{48:59} \parallel 0b0000$
 $MSR_{WE,EE,PR,IS,DS,FP,FE0,FE1} \leftarrow 0b0000_0000$

se_sc is used to request a system service. A system call interrupt is generated. The contents of the MSR are copied into SRR1 and the address of the instruction after the **se_sc** instruction is placed into SRR0.

MSR[WE,EE,PR,IS,DS,FP,FE0,FE1] are cleared.

The interrupt causes the next instruction to be fetched from the address

IVPR[32–47]||IVOR8[48–59]||0b0000

This instruction is context synchronizing.

Special Registers Altered: SRR0 SRR1 MSR[WE,EE,PR,IS,DS,FP,FE0,FE1]

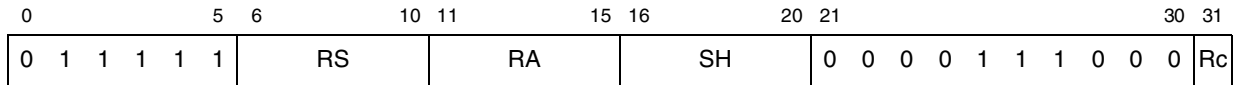
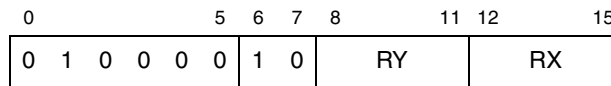
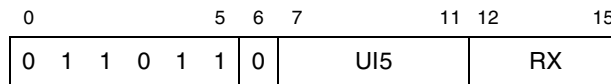
_slwx

VLE	User
-----	------

_slwx

Shift Left Word [Immediate] [and Record]

e_slwi **rA,rS,SH** (Rc = 0)
e_slwi. **rA,rS,SH** (Rc = 1)

**se_slw** **rX,rY****se_slwi** **rX,UI5**

```

if 'e_slwi[.]' then n ← SH
if se_slw then n ← GPR(RY)58:63
if se_slwi then n ← UI5
r ← ROTL32(GPR(RS or RX)32:63,n)
if n<32 then m ← MASK(32,63-n)
else m ← 320
result32:63 ← r & m
if Rc=1 then do
    LT ← result32:63 < 0
    GT ← result32:63 > 0
    EQ ← result32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RA or RX) ← result32:63

```

Let the shift count n be the value specified by the contents of bits 58–63 of GPR(**rB** or **rY**), or by the value of the SH or UI5 field.

The contents of bits 32–63 of GPR(**rS** or **rX**) are shifted left n bits. Bits shifted out of position 32 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into bits 32–63 of GPR(**rA** or **rX**).

Shift amounts from 32 to 63 give a zero result.

Special Registers Altered: CR0 (if Rc = 1)

_srawX

VLE	User
-----	------

_srawX

Shift Right Algebraic Word [Immediate] [and Record]

se_sraw **rX,rY**

0	5	6	7	8	11	12	15
0	1	0	0	0	0	0	1
						RY	RX

se_srawi **rX,UI5**

0	5	6	7	11	12	15	
0	1	1	0	1	0	1	
						UI5	RX

```

if `se_sraw` then n ← GPR(RY)59:63
if `se_srawi` then n ← UI5
r ← ROTL32(GPR(RS or RX)32:63, 32-n)
if ((se_sraw & GPR(RY)58=1) then m ← 320
else m ← MASK(n+32, 63)
s ← GPR(RS or RX)32
result0:63 ← r&m | (32s)&~m
if Rc=1 then do
    LT ← result32:63 < 0
    GT ← result32:63 > 0
    EQ ← result32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RA or RX) ← result32:63
CA ← s & ((r&~m)32:63≠0)

```

If **se_sraw**, let the shift count n be the contents of bits 58–63 of GPR(**rY**).

If **se_srawi**, let the shift count n be the value of the UI5 field.

The contents of bits 32–63 of GPR(**rS** or **rX**) are shifted right n bits. Bits shifted out of position 63 are lost. Bit 32 of **rS** or **rX** is replicated to fill vacated positions on the left. The 32-bit result is placed into bits 32–63 of GPR(**rA** or **rX**).

CA is set if bits 32–63 of GPR(**rS** or **rX**) contain a negative value and any 1 bits are shifted out of bit position 63; otherwise CA is cleared.

A shift amount of zero causes GPR(**rA** or **rX**) to receive EXTS(GPR(**rS** or **rX**)_{32:63}), and CA to be cleared. For **se_sraw**, shift amounts from 32 to 63 give a result of 64 sign bits and cause CA to receive bit 32 of the contents of GPR(**rS** or **rX**) (that is, sign bit of GPR(**rS** or **rX**)_{32:63}).

Special Registers Altered: CA

CR0 (if Rc = 1)

_stbx

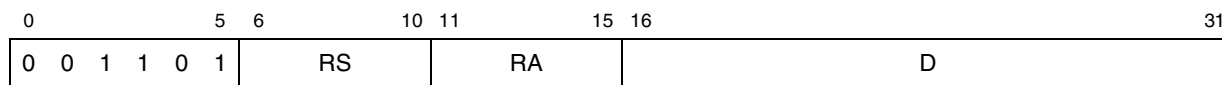
VLE	User
-----	------

_stbx

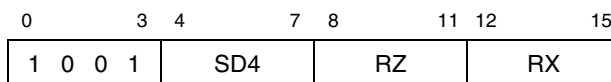
Store Byte [with Update] [Indexed]

e_stb**rS,D(rA)**

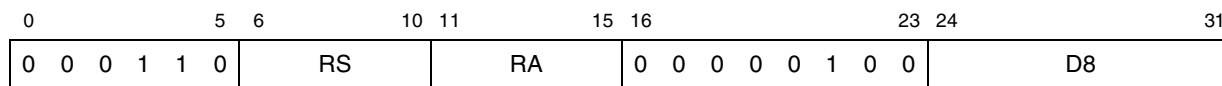
(D-mode)

**se_stb****rZ,SD4(rX)**

(SD4-mode)

**e_stbu****rS,D8(rA)**

(D8-mode)



```

if (RA=0 & !se_stb) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (280 || SD4))32:63
MEM(EA,1) ← GPR(RS or RZ)56:63
if e_stbu then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_stb** and **e_stbu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_stb**, let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field.

The contents of bits 56–63 of GPR(**rS**) are stored into the byte in memory addressed by EA.

- If **e_stbu**, EA is placed into GPR(**rA**).
- If **e_stbu** and **rA** = 0, the instruction form is invalid.
- None

_sthx

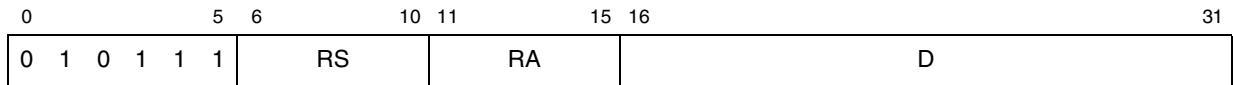
VLE	User
-----	------

_sthx

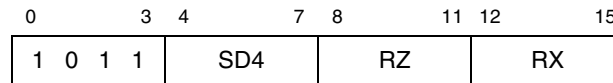
Store Halfword [with Update] [Indexed]

e_sth**rS,D(rA)**

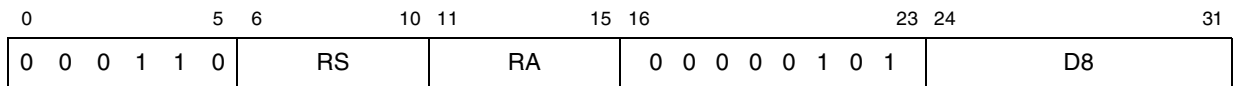
(D-mode)

**se_sth****rZ,SD4(rX)**

(SD4-mode)

**e_sthu****rS,D8(rA)**

(D8-mode)



```

if (RA=0 & !se_sth) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (270 || SD4 || 0))32:63
MEM(EA,2) ← GPR(RS or RZ)48:63
if e_sthu then GPR(RA) ← EA

```

Let the EA be calculated as follows:

- For **e_sth** and **e_sthu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_sth**, let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field shifted left by 1 bit.

The contents of bits 48–63 of GPR(**rS**) are stored into the half word in memory addressed by EA.If **e_sthu**, EA is placed into GPR(**rA**).If **e_sthu** and **rA** = 0, the instruction form is invalid.

Special Registers Altered: None

_stmw

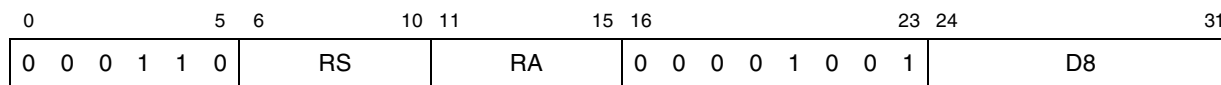
VLE	User
-----	------

_stmw

Store Multiple Word

e_stmw**rS,D8(rA)**

(D8-mode)



```

if RA=0 then EA ← EXTS(D8)32:63
else      EA ← (GPR(RA)+EXTS(D8))32:63
r ← RS
do while r ≤ 31
    MEM(EA,4) ← GPR(r)32:63
    r ← r + 1
    EA ← (EA+4)32:63

```

Let the EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D8 instruction field.

Let $n = (32 - \mathbf{rS})$. Bits 32–63 of registers GPR(**rS**) through GPR(31) are stored in n consecutive words in memory starting at address EA.

EA must be a multiple of 4. If it is not, either an alignment interrupt is invoked or the results are boundedly undefined.

Special Registers Altered: None

_stwX

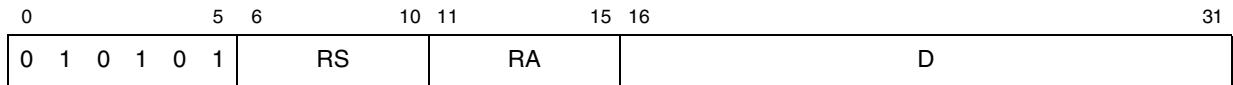
VLE	User
-----	------

_stwX

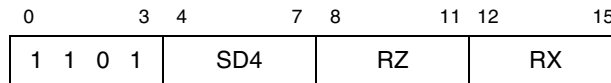
Store Word [with Update] [Indexed]

e_stw**rS,D(rA)**

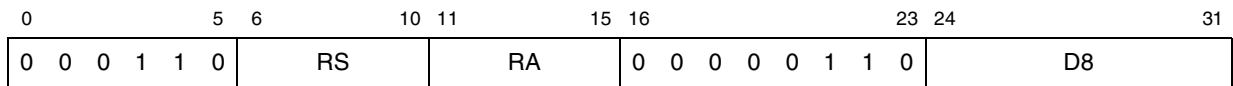
(D-mode)

**se_stw****rZ,SD4(rX)**

(SD4-mode)

**e_stwu****rS,D8(rA)**

(D8-mode)



```

if (RA=0 & !se_stw) then a ← 320 else a ← GPR(RA or RX)
if D-mode then EA ← (a + EXTS(D))32:63
if D8-mode then EA ← (a + EXTS(D8))32:63
if SD4-mode then EA ← (a + (260 || SD4 || 20))32:63
MEM(EA, 4) ← GPR(RS or RZ)32:63

```

Let the EA be calculated as follows:

- For **e_stw** and **e_stwu**, let EA be the sum of the contents of GPR(**rA**), or 32 0s if **rA** = 0, and the sign-extended value of the D or D8 instruction field.
- For **se_stw**, let EA be the sum of the contents of GPR(**rX**) and the zero-extended value of the SD4 instruction field shifted left by 2 bits.

The contents of bits 32–63 of GPR(**rS**) are stored into the word in memory addressed by EA.If **e_stwu**, EA is placed into GPR(**rA**).If **e_stwu** and **rA** = 0, the instruction form is invalid.

Special Registers Altered: None

_sub

VLE	User
-----	------

_sub

Subtract

se_sub**rX,rY**

0	5	6	7	8	11	12	15	
0	0	0	0	0	1	1	0	
					RY		RX	

$$\text{sum}_{32:63} \leftarrow \text{GPR}(\text{RX}) + \neg\text{GPR}(\text{RY}) + 1$$

$$\text{GPR}(\text{RX}) \leftarrow \text{sum}_{32:63}$$

The sum of the contents of GPR(**rX**), the one's complement of contents of GPR(**rY**), and 1 is placed into GPR(**rX**).

Special Registers Altered: None

_subfx

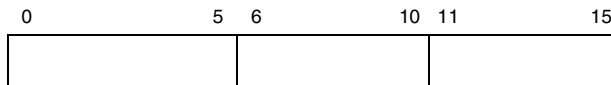
VLE	User
-----	------

_subfx

Subtract From

se_subf

rX,rY



$$\text{sum}_{32:63} \leftarrow \neg\text{GPR}(\text{RX}) + \text{GPR}(\text{RY}) + 1$$

$$\text{GPR}(\text{RX}) \leftarrow \text{sum}_{32:63}$$

The sum of the one's complement of the contents of GPR(**rX**), the contents of GPR(**rY**), and 1 is placed into GPR(**rX**).

Special Registers Altered: None

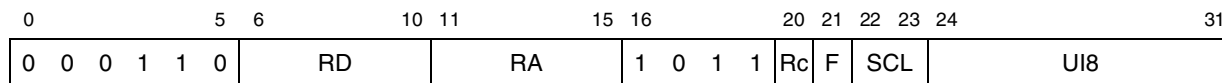
_subfix

VLE	User
-----	------

_subfix

Subtract From Immediate Carrying [and Record]

e_subfic	rD,rA,SCI8	(Rc = 0)
e_subfic.	rD,rA,SCI8	(Rc = 1)



```

imm ← SCI8(F,SCL,UI8)
carry32:63 ← Carry(¬GPR(RA) + imm + 1)
sum32:63 ← ¬GPR(RA) + imm + 1
if Rc=1 then do
    LT ← sum32:63 < 0
    GT ← sum32:63 > 0
    EQ ← sum32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RD) ← sum32:63
CA ← carry32

```

The sum of the one's complement of the contents of GPR(**rA**), the value of SCI8, and 1 is placed into GPR(**rD**).

Special Registers Altered: CA CR0 (if Rc=1)

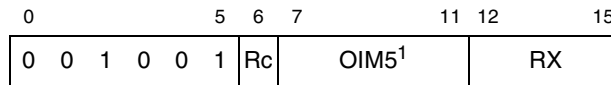
_subix

VLE	User
-----	------

_subix

Subtract Immediate [and Record]

se_subi **rX,OIMM** (Rc = 0)
se_subi. **rX,OIMM** (Rc = 1)

¹ OIMM = OIM5 + 1

```

sum32:63 ← GPR(RX) + -(270 || OFFSET(OIM5)) + 1
if Rc=1 then do
    LT ← sum32:63 < 0
    GT ← sum32:63 > 0
    EQ ← sum32:63 = 0
    CR0 ← LT || GT || EQ || SO
GPR(RX) ← sum32:63

```

The sum of the contents of GPR(**rX**), the one's complement of the zero-extended value of the offset OIM5 field (a final value in the range 1–32), and 1 is placed into GPR(**rX**).

Special Registers Altered: CR0 (if Rc = 1)

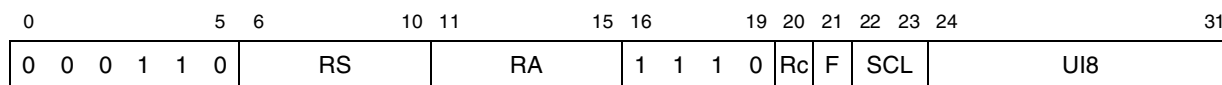
_xorX

VLE	User
-----	------

_xorX

XOR [Immediate] [and Record]

e_xori	rA,rS,SCI8	(Rc = 0)
e_xori.	rA,rS,SCI8	(Rc = 1)



```

if `e_xori[.]` then b ← SCI8(F,SCL,UI8)
result32:63 ← GPR(RS) ⊕ b
if Rc=1 then do
  LT ← result32:63 < 0
  GT ← result32:63 > 0
  EQ ← result32:63 = 0
  CR0 ← LT || GT || EQ || SO
GPR(RA) ← result

```

For **e_xori[.]**, the contents of GPR(**rS**) are XORed with SCI8.

The result is placed into GPR(**rA**).

Special Registers Altered: CR0 (if Rc = 1)

Appendix A

VLE Instruction Formats

The format diagrams in this appendix show all valid combinations of instruction fields for those formats unique to VLE-defined instructions. Instruction forms that are available in VLE or non-VLE mode are described in the EREF.

A.1 Overview

VLE instructions may be 2 or 4 bytes long and are half-word-aligned in memory. Thus, whenever instruction addresses are presented to the processor (as in branch instructions), the low-order bit is treated as 0. Similarly, whenever the processor generates an instruction address, the low-order bit is 0.

In some cases, an instruction field must contain a particular value; otherwise, the instruction form is invalid and the results are as described for invalid instruction forms in the UISA.

VLE instructions use split-field notation as defined in the instruction formats appendix of the EREF.

A.2 VLE Instruction Formats

All VLE instructions to be executed are either 2 or 4 bytes long and are halfword-aligned in memory. Thus, whenever instruction addresses are presented to the processor (as in branch instructions), the low-order bit is treated as 0. Similarly, whenever the processor generates an instruction address, the low-order bit is zero.

The format diagrams show all valid combinations of instruction fields. Only those formats unique to VLE-defined instructions are included. Instruction forms that are available in VLE or non-VLE mode are described in the EREF and are not repeated here.

In some cases, an instruction field must contain a particular value. If it does not, the instruction form is invalid and the results are as described for invalid instruction forms in the UISA.

VLE instructions use split-field notation as described in the EREF.

A.2.1 Instruction Fields

In addition to the VLE-defined instruction fields described in [Table A-1](#), VLE uses instruction fields described in the EREF.

Table A-1. Instruction Fields

Field	Description
ARX (12:15)	Field used to specify an “alternate” GPR in the range GPR8–GPR23 to be used as a destination.
ARY (8:11)	Field used to specify an “alternate” GPR in the range GPR8–GPR23 to be used as a source.

Table A-1. Instruction Fields (continued)

Field	Description
BD8 (8:15), BD15 (16:30), BD24 (7:30)	Immediate field specifying a signed two's complement branch displacement concatenated on the right with 0 and sign-extended to 64 bits. BD15 (Used by 32-bit branch conditional class instructions) A 15-bit signed displacement sign-extended and shifted left one bit (concatenated with 0) and added to the current instruction address to form the branch target address. BD24 (Used by 32-bit branch class instructions) A 24-bit signed displacement, sign-extended and shifted left one bit (concatenated with 0) and added to the current instruction address to form the branch target address. BD8 (Used by 16-bit branch and branch conditional class instructions) An 8-bit signed displacement sign-extended and shifted left one bit (concatenated with 0) and added to the current instruction address to form the branch target address.
BI16 (6:7), BI32 (12:15)	Field used to specify one of the CR fields to be used as a condition of a Branch Conditional instruction.
BO16 (5), BO32 (10:11)	Field used to specify whether to branch if the condition is true, false, or to decrement the CTR and branch if the CTR is not zero in a Branch Conditional instruction.
BF32 (9:10)	Field used to specify one of the CR fields to be used as a target of a compare instruction.
D8 (24:31)	The D8 field is a 8-bit signed displacement sign-extended to 64 bits.
F (21)	Fill value used to fill the remaining 56 bits of a scaled-immediate 8 value.
LI20 (17:20 11:15 21:31)	A 20-bit signed immediate value sign-extended to 64 bits for the <code>e_li</code> instruction.
LK (7, 16, 31)	LINK bit. 0 Do not set the LR. 1 Set the LR. The sum of the value 2 or 4 and the address of the branch instruction is placed into the LR.
OIM5 (7:11)	Offset Immediate field used to specify a 5-bit unsigned Integer value in the range [1:32] encoded as [0:31]. Thus the binary encoding of 00000 represents an immediate value of 1, 00001 represents an immediate value of 2, and so on.
OPCD(0:3, 0:4, 0:5, 0:9, 0:14, 0:15)	Primary opcode field.
Rc (6, 7, 20, 31)	RECORD bit. 0 Do not alter the CR. 1 Set CR field 0.
RX (12:15)	Specifies a GPR in the ranges GPR0–GPR7 or GPR24–GPR31 to be used as a source or as a destination. R0 is encoded as 0000, R1 as 0001... R24 as 1000, R25 as 1001, etc.
RY (8:11)	Specifies a GPR in the ranges GPR0–GPR7 or GPR24–GPR31 to be used as a source. R0 is encoded as 0000, R1 as 0001... R24 is as 1000, R25 as 1001, etc.
RZ (8:11)	Specifies a GPR in the ranges GPR0–GPR7 or GPR24–GPR31 to be used as a source or as a destination for load/store data. R0 is encoded as 0000, R1 as 0001... R24 is as 1000, R25 as 1001, etc.
SCL (22:23)	Specifies a scale amount in SCI8-form Immediate instructions. Scaling involves left shifting by 0, 8, 16, or 24 bits.
SD4 (4:7)	Used by 16-bit load and store instructions. A 4-bit unsigned immediate value zero-extended to 64 bits, shifted left according to the size of the operation, and added to the base register to form a 64-bit EA. For byte operations, no shift is performed. For half-word operations, the immediate is shifted left one bit (concatenated with 0). For word operations, the immediate is shifted left two bits (concatenated with '00).

Table A-1. Instruction Fields (continued)

Field	Description
SI (6:10 21:31, 11:15 21:31)	A 16-bit signed immediate value sign-extended to 64 bits and used as one operand of the instruction.
UI (6:10 21:31, 11:15 21:31)	A 16-bit unsigned immediate value zero-extended to 64 bits or padded with 16 zeros and used as one operand of the instruction. The instruction encoding differs between the I16A and I16L instruction formats as shown in Section A.13, "I16A Form," and Section A.14, "I16L Form."
UI5 (7:11)	Immediate field used to specify a 5-bit unsigned Integer value.
UI7 (5:11)	Immediate field used to specify a 7-bit unsigned Integer value.
UI8 (24:31)	Immediate field used to specify an 8-bit unsigned Integer value.
XO (6, 6:7, 6:10, 6:11, 16, 16:19, 16:23)	Extended opcode field.

NOTE (Programming)

For scaled immediate instructions using the SCI8-form, the instruction assembly syntax requires a single immediate value, `sci8`, that the assembler synthesizes into the appropriate F, SCL, and UI8 fields. The F, SCL, and UI8 fields must be able to be formed correctly from the given `sci8` value or the assembler flags the assembly instruction as an error.

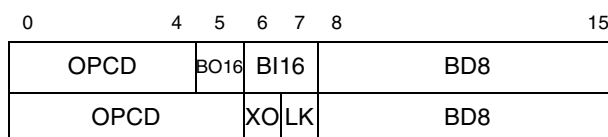
A.2.2 BD8 Form (16-Bit Branch Instructions)

Figure A-1. BD8 Instruction Format

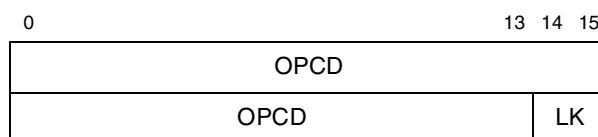
A.3 C Form (16-Bit Control Instructions)

Figure A-2. C Instruction Format

A.4 IM5 Form (16-Bit register + immediate Instructions)

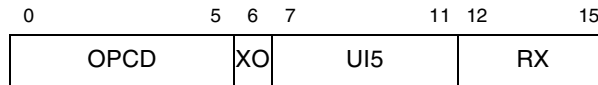


Figure A-3. IM5 Instruction Format

A.5 OIM5 Form (16-Bit Register + Offset Immediate Instructions)

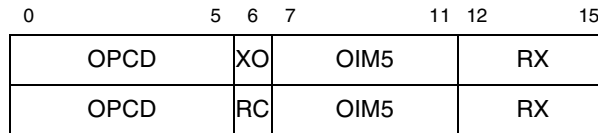


Figure A-4. OIM5 Instruction Format

A.6 IM7 Form (16-Bit Load immediate Instructions)

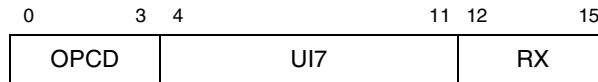


Figure A-5. IM7 Instruction Format

A.7 R Form (16-Bit Monadic Instructions)

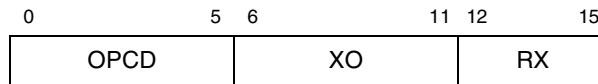


Figure A-6. R Instruction Format

A.8 RR Form (16-Bit Dyadic Instructions)

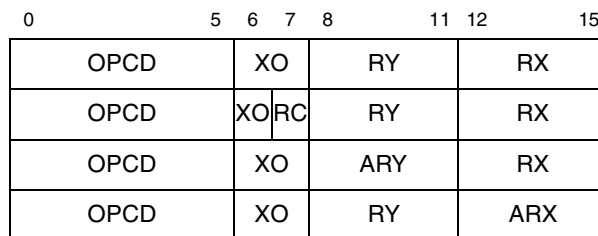


Figure A-7. RR Instruction Format

A.9 SD4 Form (16-Bit Load/Store Instructions)

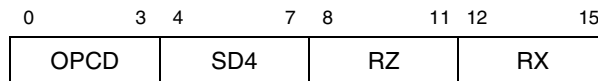


Figure A-8. SD4 Instruction Format

A.10 BD15 Form

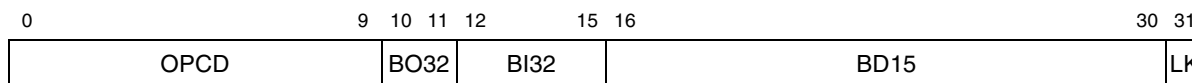


Figure A-9. BD15 Instruction Format

A.11 BD24 Form

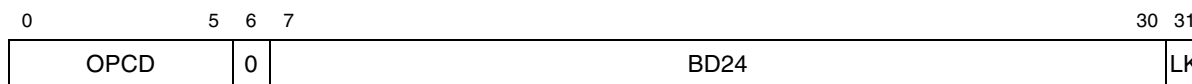


Figure A-10. BD24 Instruction Format

A.12 D8 Form

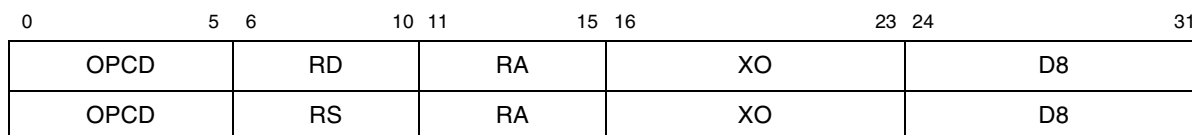


Figure A-11. D8 Instruction Format

A.13 I16A Form

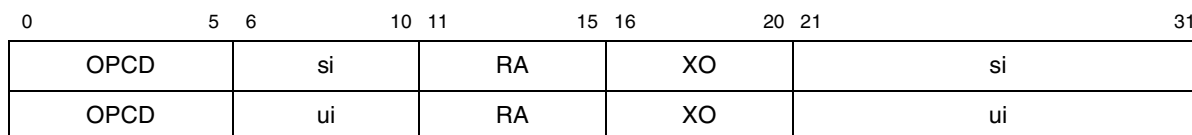


Figure A-12. I16A Instruction Format

A.14 I16L Form

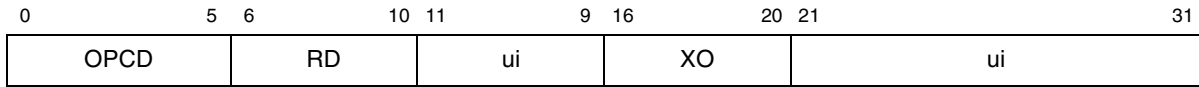


Figure A-13. I16L Instruction Format

A.15 M Form

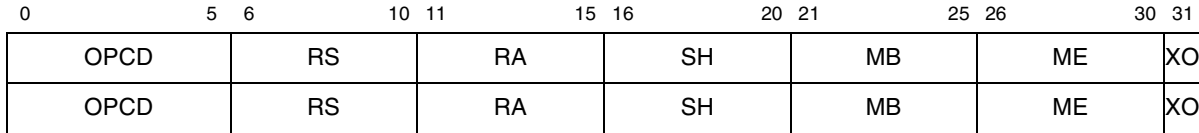


Figure A-14. M Instruction Format

A.16 SCI8 Form

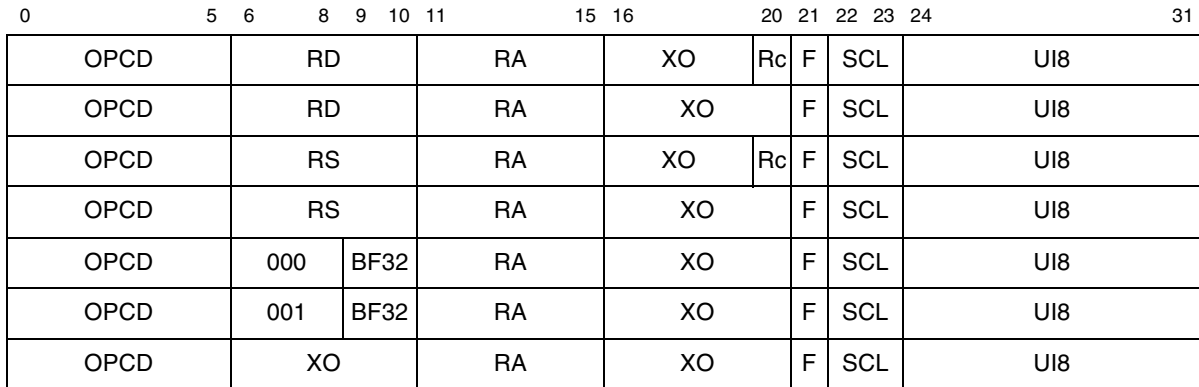


Figure A-15. SC18 Instruction Format

A.17 LI20 Form

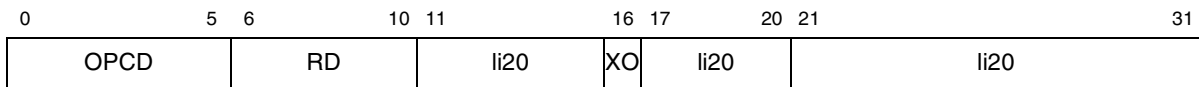


Figure A-16. LI20 Instruction Format

Appendix B

VLE Instruction Set Tables

This appendix lists VLE-supported instructions by mnemonic and by opcode.

Except as described below and in the “Effective Address Calculation” section of the EREF, all instructions are independent of whether the processor is in 32- or 64-bit mode.

Table B-1. Mode Dependency and Privilege Abbreviations

Abbreviation	Description
Mode Dependency Column	
CT	If the instruction tests the CTR, it tests the low-order 32 bits in 32-bit mode and all 64 bits in 64-bit mode.
SR	The setting of status registers (such as XER and CR0) is mode-dependent.
32	The instruction must be executed only in 32-bit mode.
64	The instruction must be executed only in 64-bit mode.
Privilege Column	
P	Denotes a privileged instruction.
O	Denotes an instruction that is treated as privileged or nonprivileged, depending on the SPR number.
M	Denotes an instruction that is treated as privileged or nonprivileged, depending on the value of the MSR[UCLE].

B.1 VLE Instruction Set Sorted by Mnemonic

Table B-2 lists all instructions available in VLE mode in the Power ISA, in order by mnemonic. Opcodes not listed below are treated as illegal by VLE.

Table B-2. VLE Instruction Set Sorted by Mnemonic

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
XO	7C000214			B	add[o][.]	Add
XO	7C000014			B	addc[o][.]	Add Carrying
XO	7C000114	SR		B	adde[o][.]	Add Extended
XO	7C0001D4	SR		B	addme[o][.]	Add to Minus One Extended
XO	7C000194	SR		B	addze[o][.]	Add to Zero Extended
X	7C000038	SR		B	and[.]	AND
X	7C000078	SR		B	andc[.]	AND with Complement
EVX	1000020F			SP	brinc	Bit Reverse Increment

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C000000			B	cmp	Compare
X	7C000040			B	cmpl	Compare Logical
X	7C000074	SR		64	cntlzd[.]	Count Leading Zeros Doubleword
X	7C000034	SR		B	cntlzw[.]	Count Leading Zeros Word
X	7C0005EC			E	dcba	Data Cache Block Allocate
X	7C0000AC			B	dcbf	Data Cache Block Flush
X	7C0000FE		P	E.PD	dcbfep	Data Cache Block Flush by External Process ID
X	7C0003AC		P	E	dcbi	Data Cache Block Invalidate
X	7C00030C		M	E.CL	dcblc	Data Cache Block Lock Clear
X	7C00006C			B	dcbst	Data Cache Block Store
X	7C00022C			B	dcbt	Data Cache Block Touch
X	7C00027E		P	E.PD	dcbtstep	Data Cache Block Touch by External Process ID
X	7C00014C		M	E.CL	dcbtIs	Data Cache Block Touch and Lock Set
X	7C0001EC			B	dcbtst	Data Cache Block Touch for Store
X	7C0001FE		P	E.PD	dcbtstep	Data Cache Block Touch for Store by External Process ID
X	7C00010C		M	E.CL	dcbtstIs	Data Cache Block Touch for Store and Lock Set
X	7C0007EC			B	dcbz	Data Cache Block set to Zero
X	7C0007FE		P	E.PD	dcbzep	Data Cache Block set to Zero by External Process ID
X	7C00038C		P	E.CI	dci	Data Cache Invalidate
X	7C00028C		P	E.CD	dcread	Data Cache Read
X	7C0003CC		P	E.CD	dcread	Data Cache Read
XO	7C0003D2	SR		64	divd[o][.]	Divide Doubleword
XO	7C000392	SR		64	divdu[o][.]	Divide Doubleword Unsigned
XO	7C0003D6	SR		B	divw[o][.]	Divide Word
XO	7C000396	SR		B	divwu[o][.]	Divide Word Unsigned
D	1C000000			VLE	e_add16i	Add Immediate
I16A	70008800	SR		VLE	e_add2i.	Add (2 operand) Immediate and Record
I16A	70009000			VLE	e_add2is	Add (2 operand) Immediate Shifted
SCI8	18008000	SR		VLE	e_addi[.]	Add Scaled Immediate
SCI8	18009000	SR		VLE	e_addic[.]	Add Scaled Immediate Carrying
I16L	7000C800	SR		VLE	e_and2i.	AND (2 operand) Immediate

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
I16L	7000E800	SR		VLE	e_and2is.	AND (2 operand) Immediate Shifted
SCI8	1800C000	SR		VLE	e_andi[.]	AND Scaled Immediate
BD24	78000000			VLE	e_b[l]	Branch [and Link]
BD15	7A000000	CT		VLE	e_bc[l]	Branch Conditional [and Link]
IA16	70009800			VLE	e_cmp16i	Compare Immediate Word
IA16	7000B000			VLE	e_cmph16i	Compare Halfword Immediate
X	7C00001C			VLE	e_cmph	Compare Halfword
IA16	7000B800			VLE	e_cmphl16i	Compare Halfword Logical Immediate
X	7C00005C			VLE	e_cmphl	Compare Halfword Logical
SCI8	1800A800			VLE	e_cmpi	Compare Scaled Immediate Word
I16A	7000A800			VLE	e_cmpl16i	Compare Logical Immediate Word
SCI8	1880A800			VLE	e_cmpli	Compare Logical Scaled Immediate Word
XL	7C000202			VLE	e_crand	Condition Register AND
XL	7C000102			VLE	e_crandc	Condition Register AND with Complement
XL	7C000242			VLE	e_creqv	Condition Register Equivalent
XL	7C0001C2			VLE	e_crnand	Condition Register NAND
XL	7C000042			VLE	e_crnor	Condition Register NOR
XL	7C000382			VLE	e_cror	Condition Register OR
XL	7C000342			VLE	e_crorc	Condition Register OR with Complement
XL	7C000182			VLE	e_crxor	Condition Register XOR
D	30000000			VLE	e_lbz	Load Byte and Zero
D8	18000000			VLE	e_lbzu	Load Byte and Zero with Update
D	38000000			VLE	e_lha	Load Halfword Algebraic
D8	18000300			VLE	e_lhau	Load Halfword Algebraic with Update
D	58000000			VLE	e_lhz	Load Halfword and Zero
D8	18000100			VLE	e_lhzu	Load Halfword and Zero with Update
LI20	70000000			VLE	e_li	Load Immediate
I16L	7000E000			VLE	e_lis	Load Immediate Shifted
D8	18000800			VLE	e_lmww	Load Multiple Word
D	50000000			VLE	e_lwz	Load Word and Zero
D8	18000200			VLE	e_lwzu	Load Word and Zero with Update
XL	7C000020			VLE	e_mcrf	Move CR Field

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
I16A	7000A000			VLE	e_mull2i	Multiply (2 operand) Low Immediate
SCI8	1800A000			VLE	e_mulli	Multiply Low Scaled Immediate
I16L	7000C000			VLE	e_or2i	OR (2operand) Immediate
I16L	7000D000			VLE	e_or2is	OR (2 operand) Immediate Shifted
SCI8	1800D000	SR		VLE	e_ori[.]	OR Scaled Immediate
X	7C000230	SR		VLE	e_rlwi[.]	Rotate Left Word
X	7C000270	SR		VLE	e_rlwi[.]	Rotate Left Word Immediate
M	74000000			VLE	e_rlwimi	Rotate Left Word Immediate then Mask Insert
M	74000001			VLE	e_rlwinm	Rotate Left Word Immediate then AND with Mask
X	7C000070	SR		VLE	e_slwi[.]	Shift Left Word Immediate
X	7C000470	SR		VLE	e_srwi[.]	Shift Right Word Immediate
D	34000000			VLE	e_stb	Store Byte
D8	18000400			VLE	e_stbu	Store Byte with Update
D	5C000000			VLE	e_sth	Store Halfword
D8	18000500			VLE	e_sthu	Store Halfword with Update
D8	18000900			VLE	e_stmw	Store Multiple Word
D	54000000			VLE	e_stw	Store Word
D8	18000600			VLE	e_stwu	Store word with Update
SCI8	1800B000	SR		VLE	e_subfic[.]	Subtract From Scaled Immediate Carrying
SCI8	1800E000	SR		VLE	e_xori[.]	XOR Scaled Immediate
EVX	100002E4			SP.FD	efdabs	Floating-Point Double-Precision Absolute Value
EVX	100002E0			SP.FD	efdadd	Floating-Point Double-Precision Add
EVX	100002EF			SP.FD	efdcfs	Floating-Point Double-Precision Convert from Single-Precision
EVX	100002F3			SP.FD	efdcfsf	Convert Floating-Point Double-Precision from Signed Fraction
EVX	100002F1			SP.FD	efdcfsi	Convert Floating-Point Double-Precision from Signed Integer
EVX	100002E3			SP.FD	efdcfsid	Convert Floating-Point Double-Precision from Signed Integer Doubleword
EVX	100002F2			SP.FD	efdcfuf	Convert Floating-Point Double-Precision from Unsigned Fraction
EVX	100002F0			SP.FD	efdcfui	Convert Floating-Point Double-Precision from Unsigned Integer

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100002E2			SP.FD	efdcfluid	Convert Floating-Point Double-Precision from Unsigned Integer Doubleword
EVX	100002EE			SP.FD	efdcmeq	Floating-Point Double-Precision Compare Equal
EVX	100002EC			SP.FD	efdcmpgt	Floating-Point Double-Precision Compare Greater Than
EVX	100002ED			SP.FD	efdcmplt	Floating-Point Double-Precision Compare Less Than
EVX	100002F7			SP.FD	efdcstf	Convert Floating-Point Double-Precision to Signed Fraction
EVX	100002F5			SP.FD	efdcstsi	Convert Floating-Point Double-Precision to Signed Integer
EVX	100002EB			SP.FD	efdcstsidz	Convert Floating-Point Double-Precision to Signed Integer Doubleword with Round Towards Zero
EVX	100002FA			SP.FD	efdcstsiz	Convert Floating-Point Double-Precision to Signed Integer with Round Towards Zero
EVX	100002F6			SP.FD	efdcstuf	Convert Floating-Point Double-Precision to Unsigned Fraction
EVX	100002F4			SP.FD	efdcstui	Convert Floating-Point Double-Precision to Unsigned Integer
EVX	100002EA			SP.FD	efdcstuidz	Convert Floating-Point Double-Precision to Unsigned Integer Doubleword with Round Towards Zero
EVX	100002F8			SP.FD	efdcstuiiz	Convert Floating-Point Double-Precision to Unsigned Integer with Round Towards Zero
EVX	100002E9			SP.FD	efddiv	Floating-Point Double-Precision Divide
EVX	100002E8			SP.FD	efdmul	Floating-Point Double-Precision Multiply
EVX	100002E5			SP.FD	efdnabs	Floating-Point Double-Precision Negative Absolute Value
EVX	100002E6			SP.FD	efdneg	Floating-Point Double-Precision Negate
EVX	100002E1			SP.FD	efdsb	Floating-Point Double-Precision Subtract
EVX	100002FE			SP.FD	efdsteq	Floating-Point Double-Precision Test Equal
EVX	100002FC			SP.FD	efdstgt	Floating-Point Double-Precision Test Greater Than
EVX	100002FD			SP.FD	efdstlt	Floating-Point Double-Precision Test Less Than
EVX	100002E4			SP.FS	efsabs	Floating-Point Single-Precision Absolute Value
EVX	100002E0			SP.FS	efsadd	Floating-Point Single-Precision Add
EVX	100002CF			SP.FD	efscfd	Floating-Point Single-Precision Convert from Double-Precision
EVX	100002F3			SP.FS	efscfsf	Convert Floating-Point Single-Precision from Signed Fraction

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100002F1			SP.FS	efscfsi	Convert Floating-Point Single-Precision from Signed Integer
EVX	100002E3			SP.FS	efscfsid	Convert Floating-Point Single-Precision from Signed Integer Doubleword
EVX	100002F2			SP.FS	efscfuf	Convert Floating-Point Single-Precision from Unsigned Fraction
EVX	100002F0			SP.FS	efscfui	Convert Floating-Point Single-Precision from Unsigned Integer
EVX	100002E2			SP.FS	efscfuid	Convert Floating-Point Single-Precision from Unsigned Integer Doubleword
EVX	100002EE			SP.FS	efscmpeq	Floating-Point Single-Precision Compare Equal
EVX	100002EC			SP.FS	efscmpgt	Floating-Point Single-Precision Compare Greater Than
EVX	100002ED			SP.FS	efscmplt	Floating-Point Single-Precision Compare Less Than
EVX	100002F7			SP.FS	efscstf	Convert Floating-Point Single-Precision to Signed Fraction
EVX	100002F5			SP.FS	efscstsi	Convert Floating-Point Single-Precision to Signed Integer
EVX	100002EB			SP.FS	efscstsidz	Convert Floating-Point Single-Precision to Signed Integer Doubleword with Round Towards Zero
EVX	100002FA			SP.FS	efscstsiz	Convert Floating-Point Single-Precision to Signed Integer with Round Towards Zero
EVX	100002F6			SP.FS	efscstuf	Convert Floating-Point Single-Precision to Unsigned Fraction
EVX	100002F4			SP.FS	efscstui	Convert Floating-Point Single-Precision to Unsigned Integer
EVX	100002EA			SP.FS	efscstuidz	Convert Floating-Point Single-Precision to Unsigned Integer Doubleword with Round Towards Zero
EVX	100002F8			SP.FS	efscstuiiz	Convert Floating-Point Single-Precision to Unsigned Integer with Round Towards Zero
EVX	100002E9			SP.FS	efscdiv	Floating-Point Single-Precision Divide
EVX	100002E8			SP.FS	efscmul	Floating-Point Single-Precision Multiply
EVX	100002E5			SP.FS	efscnabs	Floating-Point Single-Precision Negative Absolute Value
EVX	100002E6			SP.FS	efscneg	Floating-Point Single-Precision Negate
EVX	100002E1			SP.FS	efscsub	Floating-Point Single-Precision Subtract
EVX	100002FE			SP.FS	efcststeq	Floating-Point Single-Precision Test Equal
EVX	100002FC			SP.FS	efcststgt	Floating-Point Single-Precision Test Greater Than
EVX	100002FD			SP.FS	efcststlt	Floating-Point Single-Precision Test Less Than

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C000238	SR		B	eqv[.]	Equivalent
EVX	10000208			SP	evabs	Vector Absolute Value
EVX	10000202			SP	evaddiw	Vector Add Immediate Word
EVX	100004C9			SP	evaddsmiaaw	Vector Add Signed, Modulo, Integer to Accumulator Word
EVX	100004C1			SP	evaddssiaaw	Vector Add Signed, Saturate, Integer to Accumulator Word
EVX	100004C8			SP	evaddumiaaw	Vector Add Unsigned, Modulo, Integer to Accumulator Word
EVX	100004C0			SP	evaddusiaaw	Vector Add Unsigned, Saturate, Integer to Accumulator Word
EVX	10000200			SP	evaddw	Vector Add Word
EVX	10000211			SP	evand	Vector AND
EVX	10000212			SP	evandc	Vector AND with Complement
EVX	10000234			SP	evcmpeq	Vector Compare Equal
EVX	10000231			SP	evcmpgts	Vector Compare Greater Than Signed
EVX	10000230			SP	evcmpgtu	Vector Compare Greater Than Unsigned
EVX	10000233			SP	evcmplt	Vector Compare Less Than Signed
EVX	10000232			SP	evcmpltu	Vector Compare Less Than Unsigned
EVX	1000020E			SP	evcntlsw	Vector Count Leading Sign Bits Word
EVX	1000020D			SP	evcntlzw	Vector Count Leading Zeros Bits Word
EVX	100004C6			SP	evdivws	Vector Divide Word Signed
EVX	100004C7			SP	evdivwu	Vector Divide Word Unsigned
EVX	10000219			SP	eveqv	Vector Equivalent
EVX	1000020A			SP	evextsb	Vector Extend Sign Byte
EVX	1000020B			SP	evextsh	Vector Extend Sign Halfword
EVX	10000284			SP.FV	evfsabs	Vector Floating-Point Single-Precision Absolute Value
EVX	10000280			SP.FV	evfsadd	Vector Floating-Point Single-Precision Add
EVX	10000293			SP.FV	evfscfsf	Vector Convert Floating-Point Single-Precision from Signed Fraction
EVX	10000291			SP.FV	evfscfsi	Vector Convert Floating-Point Single-Precision from Signed Integer
EVX	10000292			SP.FV	evfscfuf	Vector Convert Floating-Point Single-Precision from Unsigned Fraction

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	10000290			SP.FV	evfscfui	Vector Convert Floating-Point Single-Precision from Unsigned Integer
EVX	1000028E			SP.FV	evfscmpeq	Vector Floating-Point Single-Precision Compare Equal
EVX	1000028C			SP.FV	evfscmpgt	Vector Floating-Point Single-Precision Compare Greater Than
EVX	1000028D			SP.FV	evfscmplt	Vector Floating-Point Single-Precision Compare Less Than
EVX	10000297			SP.FV	evfsctsf	Vector Convert Floating-Point Single-Precision to Signed Fraction
EVX	10000295			SP.FV	evfsctsi	Vector Convert Floating-Point Single-Precision to Signed Integer
EVX	1000029A			SP.FV	evfsctsiz	Vector Convert Floating-Point Single-Precision to Signed Integer with Round Towards Zero
EVX	10000296			SP.FV	evfsctuf	Vector Convert Floating-Point Single-Precision to Unsigned Fraction
EVX	10000294			SP.FV	evfsctui	Vector Convert Floating-Point Single-Precision to Unsigned Integer
EVX	10000298			SP.FV	evfsctuiZ	Vector Convert Floating-Point Single-Precision to Unsigned Integer with Round Towards Zero
EVX	10000289			SP.FV	evfsdiv	Vector Floating-Point Single-Precision Divide
EVX	10000288			SP.FV	evfsmul	Vector Floating-Point Single-Precision Multiply
EVX	10000285			SP.FV	evfsnabs	Vector Floating-Point Single-Precision Negative Absolute Value
EVX	10000286			SP.FV	evfsneg	Vector Floating-Point Single-Precision Negate
EVX	10000281			SP.FV	evfssub	Vector Floating-Point Single-Precision Subtract
EVX	1000029E			SP.FV	evfststeq	Vector Floating-Point Single-Precision Test Equal
EVX	1000029C			SP.FV	evfststgt	Vector Floating-Point Single-Precision Test Greater Than
EVX	1000029D			SP.FV	evfststlt	Vector Floating-Point Single-Precision Test Less Than
EVX	10000301			SP	evldd	Vector Load Doubleword into Doubleword
EVX	7C00011D		P	E.PD	evlddepX	Vector Load Doubleword into Doubleword by External Process ID Indexed
EVX	10000300			SP	evlddx	Vector Load Doubleword into Doubleword Indexed
EVX	10000305			SP	evldh	Vector Load Doubleword into 4 Halfwords
EVX	10000304			SP	evldhx	Vector Load Doubleword into 4 Halfwords Indexed
EVX	10000303			SP	evldw	Vector Load Doubleword into 2 Words
EVX	10000302			SP	evldwx	Vector Load Doubleword into 2 Words Indexed

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000309			SP	evlhhesplat	Vector Load Halfword into Halfwords Even and Splat
EVX	1000308			SP	evlhhesplatx	Vector Load Halfword into Halfwords Even and Splat Indexed
EVX	100030F			SP	evlhhosplat	Vector Load Halfword into Halfwords Odd and Splat
EVX	100030E			SP	evlhhosplatx	Vector Load Halfword into Halfwords Odd Signed and Splat Indexed
EVX	100030D			SP	evlhhosplat	Vector Load Halfword into Halfwords Odd Unsigned and Splat
EVX	100030C			SP	evlhhosplatx	Vector Load Halfword into Halfwords Odd Unsigned and Splat Indexed
EVX	1000311			SP	evlwhe	Vector Load Word into Two Halfwords Even
EVX	1000310			SP	evlwhex	Vector Load Word into Two Halfwords Even Indexed
EVX	1000317			SP	evlw hos	Vector Load Word into Two Halfwords Odd Signed
EVX	1000316			SP	evlw hosx	Vector Load Word into Two Halfwords Odd Signed Indexed
EVX	1000315			SP	evlw hou	Vector Load Word into Two Halfwords Odd Unsigned
EVX	1000314			SP	evlw hou x	Vector Load Word into Two Halfwords Odd Unsigned Indexed
EVX	100031D			SP	evlw hsplat	Vector Load Word into Two Halfwords and Splat
EVX	100031C			SP	evlw hsplatx	Vector Load Word into Two Halfwords and Splat Indexed
EVX	1000319			SP	evlw wsplat	Vector Load Word into Word and Splat
EVX	1000318			SP	evlw wsplatx	Vector Load Word into Word and Splat Indexed
EVX	100022C			SP	evmergehi	Vector Merge High
EVX	100022E			SP	evmergehilo	Vector Merge High/Low
EVX	100022D			SP	evmerge lo	Vector Merge Low
EVX	100022F			SP	evmerge lohi	Vector Merge Low/High
EVX	100052B			SP	evmhegsmfaa	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Fractional and Accumulate
EVX	10005AB			SP	evmhegsmfan	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Fractional and Accumulate Negative
EVX	1000529			SP	evmhegsmiaa	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Integer and Accumulate
EVX	10005A9			SP	evmhegsmian	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Integer and Accumulate Negative
EVX	1000528			SP	evmhegumiaa	Vector Multiply Halfwords, Even, Guarded, Unsigned, Modulo, Integer and Accumulate

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100005A8			SP	evmhegumian	Vector Multiply Halfwords, Even, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
EVX	1000040B			SP	evmhesmf	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional
EVX	1000042B			SP	evmhesmf a	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional to Accumulate
EVX	1000050B			SP	evmhesmf aaw	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional and Accumulate into Words
EVX	1000058B			SP	evmhesmf anw	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	10000409			SP	evmhesmi	Vector Multiply Halfwords, Even, Signed, Modulo, Integer
EVX	10000429			SP	evmhesmi a	Vector Multiply Halfwords, Even, Signed, Modulo, Integer to Accumulator
EVX	10000509			SP	evmhesmi aaw	Vector Multiply Halfwords, Even, Signed, Modulo, Integer and Accumulate into Words
EVX	10000589			SP	evmhesmi anw	Vector Multiply Halfwords, Even, Signed, Modulo, Integer and Accumulate Negative into Words
EVX	10000403			SP	evmhessf	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional
EVX	10000423			SP	evmhessf a	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional to Accumulator
EVX	10000503			SP	evmhessf aaw	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional and Accumulate into Words
EVX	10000583			SP	evmhessf anw	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional and Accumulate Negative into Words
EVX	10000501			SP	evmhessiaaw	Vector Multiply Halfwords, Even, Signed, Saturate, Integer and Accumulate into Words
EVX	10000581			SP	evmhessianw	Vector Multiply Halfwords, Even, Signed, Saturate, Integer and Accumulate Negative into Words
EVX	10000408			SP	evmheumi	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer
EVX	10000428			SP	evmheumia	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer to Accumulator
EVX	10000508			SP	evmheumiaaw	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer and Accumulate into Words
EVX	10000588			SP	evmheumianw	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	10000500			SP	evmheusiaaw	Vector Multiply Halfwords, Even, Unsigned, Saturate Integer and Accumulate into Words

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000580			SP	evmheusianw	Vector Multiply Halfwords, Even, Unsigned, Saturate Integer and Accumulate Negative into Words
EVX	100052F			SP	evmhogsmfaa	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Fractional and Accumulate
EVX	10005AF			SP	evmhogsmfan	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Fractional and Accumulate Negative
EVX	100052D			SP	evmhogsmiaa	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Integer and Accumulate
EVX	10005AD			SP	evmhogsmian	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Integer and Accumulate Negative
EVX	100052C			SP	evmhogumiaa	Vector Multiply Halfwords, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate
EVX	10005AC			SP	evmhogumian	Vector Multiply Halfwords, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
EVX	100040F			SP	evmhosmf	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional
EVX	100042F			SP	evmhosmfa	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional to Accumulator
EVX	100050F			SP	evmhosmfaaw	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional and Accumulate into Words
EVX	100058F			SP	evmhosmfanw	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	100040D			SP	evmhosmi	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer
EVX	100042D			SP	evmhosmia	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer to Accumulator
EVX	100050D			SP	evmhosmiaaw	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer and Accumulate into Words
EVX	100058D			SP	evmhosmianw	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer and Accumulate Negative into Words
EVX	1000407			SP	evmhossf	Vector Multiply Halfwords, Odd, Signed, Fractional
EVX	1000427			SP	evmhossfa	Vector Multiply Halfwords, Odd, Signed, Fractional to Accumulator
EVX	1000507			SP	evmhossfaaw	Vector Multiply Halfwords, Odd, Signed, Saturate, Fractional and Accumulate into Words
EVX	1000587			SP	evmhossfanw	Vector Multiply Halfwords, Odd, Signed, Saturate, Fractional and Accumulate Negative into Words
EVX	1000505			SP	evmhossiaaw	Vector Multiply Halfwords, Odd, Signed, Saturate, Integer and Accumulate into Words

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	10000585			SP	evmhossianw	Vector Multiply Halfwords, Odd, Signed, Saturate, Integer and Accumulate Negative into Words
EVX	1000040C			SP	evmhoumi	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer
EVX	1000042C			SP	evmhoumia	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer to Accumulator
EVX	1000050C			SP	evmhoumiaaw	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer and Accumulate into Words
EVX	1000058C			SP	evmhoumianw	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	10000504			SP	evmhousiaaw	Vector Multiply Halfwords, Odd, Unsigned, Saturate, Integer and Accumulate into Words
EVX	10000584			SP	evmhousianw	Vector Multiply Halfwords, Odd, Unsigned, Saturate, Integer and Accumulate Negative into Words
EVX	100004C4			SP	evmra	Initialize Accumulator
EVX	1000044F			SP	evmwhsmf	Vector Multiply Word High Signed, Modulo, Fractional
EVX	1000046F			SP	evmwhsmfa	Vector Multiply Word High Signed, Modulo, Fractional to Accumulator
EVX	1000054F			SP	evmwhsmfaaw	Vector Multiply Word High Signed, Modulo, Fractional and Accumulate into Words
EVX	100005CF			SP	evmwhsmfanw	Vector Multiply Word High Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	1000044D			SP	evmwhsmi	Vector Multiply Word High Signed, Modulo, Integer
EVX	1000046D			SP	evmwhsmia	Vector Multiply Word High Signed, Modulo, Integer to Accumulator
EVX	1000054D			SP	evmwhsmiaaw	Vector Multiply Word High Signed, Modulo, Integer and Accumulate into Words
EVX	100005CD			SP	evmwhsmianw	Vector Multiply Word High Signed, Modulo, Integer and Accumulate Negative into Words
EVX	10000447			SP	evmwhssf	Vector Multiply Word High Signed, Fractional
EVX	10000467			SP	evmwhssfa	Vector Multiply Word High Signed, Fractional to Accumulator
EVX	10000547			SP	evmwhssfaaw	Vector Multiply Word High Signed, Fractional and Accumulate into Words
EVX	100005C7			SP	evmwhssfanw	Vector Multiply Word High Signed, Fractional and Accumulate Negative into Words
EVX	100005C5			SP	evmwhssianw	Vector Multiply Word High Signed, Integer and Accumulate Negative into Words
EVX	1000044C			SP	evmwhumi	Vector Multiply Word High Unsigned, Modulo, Integer

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100046C			SP	evmwhumia	Vector Multiply Word High Unsigned, Modulo, Integer to Accumulator
EVX	1000054C			SP	evmwhumiaaw	Vector Multiply Word High Unsigned, Modulo, Integer and Accumulate into Words
EVX	100005CC			SP	evmwhumianw	Vector Multiply Word High Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	10000544			SP	evmwhusiaaw	Vector Multiply Word High Unsigned, Integer and Accumulate into Words
EVX	100005C4			SP	evmwhusianw	Vector Multiply Word High Unsigned, Integer and Accumulate Negative into Words
EVX	10000549			SP	evmwlsmiaaw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate into Words
EVX	100005C9			SP	evmwlsmianw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate Negative into Words
EVX	10000541			SP	evmwlssiaaw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate into Words
EVX	100005C1			SP	evmwlssianw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate Negative into Words
EVX	10000448			SP	evmwлумi	Vector Multiply Word Low Unsigned, Modulo, Integer
EVX	10000468			SP	evmwлумia	Vector Multiply Word Low Unsigned, Modulo, Integer to Accumulator
EVX	10000548			SP	evmwлумiaaw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate into Words
EVX	100005C8			SP	evmwлумianw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	10000540			SP	evmwлумisiaaw	Vector Multiply Word Low Unsigned Saturate, Integer and Accumulate into Words
EVX	100005C0			SP	evmwлумisianw	Vector Multiply Word Low Unsigned Saturate, Integer and Accumulate Negative into Words
EVX	1000045B			SP	evmwsmf	Vector Multiply Word Signed, Modulo, Fractional
EVX	1000047B			SP	evmwsmfa	Vector Multiply Word Signed, Modulo, Fractional to Accumulator
EVX	1000055B			SP	evmwsmfaa	Vector Multiply Word Signed, Modulo, Fractional and Accumulate
EVX	100005DB			SP	evmwsmfan	Vector Multiply Word Signed, Modulo, Fractional and Accumulate Negative
EVX	10000459			SP	evmwsmi	Vector Multiply Word Signed, Modulo, Integer
EVX	10000479			SP	evmwsmia	Vector Multiply Word Signed, Modulo, Integer to Accumulator

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	10000559			SP	evmwsmiaa	Vector Multiply Word Signed, Modulo, Integer and Accumulate
EVX	100005D9			SP	evmwsmian	Vector Multiply Word Signed, Modulo, Integer and Accumulate Negative
EVX	10000453			SP	evmwssf	Vector Multiply Word Signed, Saturate, Fractional
EVX	10000473			SP	evmwssfa	Vector Multiply Word Signed, Saturate, Fractional to Accumulator
EVX	10000553			SP	evmwssfaa	Vector Multiply Word Signed, Saturate, Fractional and Accumulate
EVX	100005D3			SP	evmwssfan	Vector Multiply Word Signed, Saturate, Fractional and Accumulate Negative
EVX	10000458			SP	evmwumi	Vector Multiply Word Unsigned, Modulo, Integer
EVX	10000478			SP	evmwumia	Vector Multiply Word Unsigned, Modulo, Integer to Accumulator
EVX	10000558			SP	evmwumiaa	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate
EVX	100005D8			SP	evmwumian	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate Negative
EVX	1000021E			SP	evnand	Vector NAND
EVX	10000209			SP	evneg	Vector Negate
EVX	10000218			SP	evnor	Vector NOR
EVX	10000217			SP	evor	Vector OR
EVX	1000021B			SP	evorc	Vector OR with Complement
EVX	10000228			SP	evrlw	Vector Rotate Left Word
EVX	1000022A			SP	evrlwi	Vector Rotate Left Word Immediate
EVX	1000020C			SP	evrndw	Vector Round Word
EVSEL	10000278			SP	evsel	Vector Select
EVX	10000224			SP	evslw	Vector Shift Left Word
EVX	10000226			SP	evslwi	Vector Shift Left Word Immediate
EVX	1000022B			SP	evsplatfi	Vector Splat Fractional Immediate
EVX	10000229			SP	evsplatfi	Vector Splat Immediate
EVX	10000223			SP	evsrwis	Vector Shift Right Word Immediate Signed
EVX	10000222			SP	evsrwiu	Vector Shift Right Word Immediate Unsigned
EVX	10000221			SP	evsrws	Vector Shift Right Word Signed
EVX	10000220			SP	evsrwu	Vector Shift Right Word Unsigned

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	10000321			SP	evstd	Vector Store Doubleword of Doubleword
EVX	7C00019D		P	E.PD	evstddep	Vector Store Doubleword into Doubleword by External Process ID Indexed
EVX	10000320			SP	evstddx	Vector Store Doubleword of Doubleword Indexed
EVX	10000325			SP	evstdh	Vector Store Doubleword of Four Halfwords
EVX	10000324			SP	evstdhx	Vector Store Doubleword of Four Halfwords Indexed
EVX	10000323			SP	evstdw	Vector Store Doubleword of Two Words
EVX	10000322			SP	evstdwx	Vector Store Doubleword of Two Words Indexed
EVX	10000331			SP	evstwhe	Vector Store Word of Two Halfwords from Even
EVX	10000330			SP	evstwhex	Vector Store Word of Two Halfwords from Even Indexed
EVX	10000335			SP	evstwho	Vector Store Word of Two Halfwords from Odd
EVX	10000334			SP	evstwhox	Vector Store Word of Two Halfwords from Odd Indexed
EVX	10000339			SP	evstwwe	Vector Store Word of Word from Even
EVX	10000338			SP	evstwwe	Vector Store Word of Word from Even Indexed
EVX	1000033D			SP	evstwwo	Vector Store Word of Word from Odd
EVX	1000033C			SP	evstwwox	Vector Store Word of Word from Odd Indexed
EVX	100004CB			SP	evsubsmiaaw	Vector Subtract Signed, Modulo, Integer to Accumulator Word
EVX	100004C3			SP	evsubssiaaw	Vector Subtract Signed, Saturate, Integer to Accumulator Word
EVX	100004CA			SP	evsubfumiaaw	Vector Subtract Unsigned, Modulo, Integer to Accumulator Word
EVX	100004C2			SP	evsubfusiaaw	Vector Subtract Unsigned, Saturate, Integer to Accumulator Word
EVX	10000204			SP	evsubfw	Vector Subtract from Word
EVX	10000206			SP	evsubifw	Vector Subtract Immediate from Word
EVX	10000216			SP	evxor	Vector XOR
X	7C000774	SR		B	extsb[.]	Extend Sign Byte
X	7C000734	SR		B	extsh[.]	Extend Sign Halfword
X	7C0007B4	SR		64	extsw[.]	Extend Sign Word
X	7C0007AC			B	icbi	Instruction Cache Block Invalidate
X	7C0007BE		P	E.PD	icbiep	Instruction Cache Block Invalidate by External Process ID
X	7C0001CC		M	E.CL	icblc	Instruction Cache Block Lock Clear

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C00002C			E	icbt	Instruction Cache Block Touch
X	7C0003CC		M	E.CL	icbtl	Instruction Cache Block Touch and Lock Set
X	7C00078C		P	E.CI	ici	Instruction Cache Invalidate
X	7C0007CC		P	E.CD	icread	Instruction Cache Read
A	7C00001E			B	isel	Integer Select
X	7C0000BE		P	E.PD	lbepx	Load Byte by External Process ID Indexed
X	7C0000EE			B	lbzux	Load Byte and Zero with Update Indexed
X	7C0000AE			B	lbzx	Load Byte and Zero Indexed
X	7C0000A8			64	ldarx	Load Doubleword and Reserve Indexed
X	7C00003A		P	E.PD	ldepx	Load Doubleword by External Process ID Indexed
X	7C00006A			64	ldux	Load Doubleword with Update Indexed
X	7C00002A			64	ldx	Load Doubleword Indexed
X	7C0004BE		P	E.PD	ldfepx	Load Floating-Point Double by External Process ID Indexed
X	7C0002EE			B	lhaux	Load Halfword Algebraic with Update Indexed
X	7C0002AE			B	lhax	Load Halfword Algebraic Indexed
X	7C00062C			B	lhbrx	Load Halfword Byte-Reversed Indexed
X	7C00023E		P	E.PD	lhpepx	Load Halfword by External Process ID Indexed
X	7C00026E			B	lhzux	Load Halfword and Zero with Update Indexed
X	7C00022E			B	lhzx	Load Halfword and Zero Indexed
X	7C0004AA			MA	lswi	Load String Word Immediate
X	7C00042A			MA	lswx	Load String Word Indexed
X	7C00000E			VEC	lvebx	Load Vector Element Byte Indexed
X	7C00004E			VEC	lvehx	Load Vector Element Halfword Indexed
X	7C00024E		P	E.PD	lvepx	Load Vector by External Process ID Indexed
X	7C00020E		P	E.PD	lvepxl	Load Vector by External Process ID Indexed LRU
X	7C00008E			VEC	lviewx	Load Vector Element Word Indexed
X	7C00000C			VEC	lvsl	Load Vector for Shift Left Indexed
X	7C00004C			VEC	lvsr	Load Vector for Shift Right Indexed
X	7C0000CE			VEC	lvx[I]	Load Vector Indexed [Last]
X	7C000028			B	lwarx	Load Word and Reserve Indexed
X	7C0002EA			64	lwaux	Load Word Algebraic with Update Indexed

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C0002AA			64	lwax	Load Word Algebraic Indexed
X	7C00042C			B	lwbrx	Load Word Byte-Reversed Indexed
X	7C00003E		P	E.PD	lwepx	Load Word by External Process ID Indexed
X	7C00006E			B	lwzux	Load Word and Zero with Update Indexed
X	7C00002E			B	lwzx	Load Word and Zero Indexed
X	10000158	SR		LIM	macchw[o][.]	Multiply Accumulate Cross Halfword to Word Modulo Signed
X	100001D8	SR		LIM	macchws[o][.]	Multiply Accumulate Cross Halfword to Word Saturate Signed
X	10000198	SR		LIM	macchwsu[o][.]	Multiply Accumulate Cross Halfword to Word Saturate Unsigned
X	10000118	SR		LIM	macchwu[o][.]	Multiply Accumulate Cross Halfword to Word Modulo Unsigned
X	10000058	SR		LIM	machhw[o][.]	Multiply Accumulate High Halfword to Word Modulo Signed
X	100000D8	SR		LIM	machhws[o][.]	Multiply Accumulate High Halfword to Word Saturate Signed
X	10000098	SR		LIM	machhwsu[o][.]	Multiply Accumulate High Halfword to Word Saturate Unsigned
X	10000018	SR		LIM	machhwu[o][.]	Multiply Accumulate High Halfword to Word Modulo Unsigned
X	10000358	SR		LIM	maclhw[o][.]	Multiply Accumulate Low Halfword to Word Modulo Signed
X	100003D8	SR		LIM	maclhws[o][.]	Multiply Accumulate Low Halfword to Word Saturate Signed
X	10000398	SR		LIM	maclhwsu[o][.]	Multiply Accumulate Low Halfword to Word Saturate Unsigned
X	10000318	SR		LIM	maclhwu[o][.]	Multiply Accumulate Low Halfword to Word Modulo Unsigned
AFX	7C0006AC			E	mbar	Memory Barrier
X	7C000400			B	mcrxr	Move To Condition Register From XER
AFX	7C000026			B	mfcrr	Move From Condition Register
AFX	7C000286		P	E	mfcdrr	Move From Device Control Register
AFX	7C000246		P	E	mfcdcrux	Move From Device Control Register User Indexed
AFX	7C000206		P	E	mfcdcrx	Move From Device Control Register Indexed
X	7C0000A6		P	B	mfmsrr	Move From Machine State Register
AFX	7C100026			B	mfocrrf	Move From One Condition Register Field

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
XFX	7C00029C		O	E.PM	mfpmr	Move From Performance Monitor Register
XFX	7C0002A6		O	B	mfspr	Move From Special Purpose Register
VX	10000604			VEC	mfvscr	Move from Vector Status and Control Register
X	7C0001DC		P	E.PC	msgclr	Message Clear
X	7C00019C		P	E.PC	msgsnd	Message Send
XFX	7C000120			B	mtcrf	Move to Condition Register Fields
XFX	7C000386		P	E	mtdcr	Move To Device Control Register
X	7C000346			E	mtdcruz	Move To Device Control Register User Indexed
X	7C000306		P	E	mtdcrx	Move To Device Control Register Indexed
X	7C000124		P	E	mtmsr	Move To Machine State Register
XFX	7C100120			B	mtocrf	Move To One Condition Register Field
XFX	7C00039C		O	E.PM	mtpmr	Move To Performance Monitor Register
XFX	7C0003A6		O	B	mtspr	Move To Special Purpose Register
VX	10000644			VEC	mtvscr	Move to Vector Status and Control Register
X	10000150	SR		LIM	mulchw[o][.]	Multiply Cross Halfword to Word Signed
X	10000110	SR		LIM	mulchwu[o][.]	Multiply Cross Halfword to Word Unsigned
XO	7C000092	SR		64	mulhd[.]	Multiply High Doubleword
XO	7C000012	SR		64	mulhdu[.]	Multiply High Doubleword Unsigned
X	10000050	SR		LIM	mulhhw[o][.]	Multiply High Halfword to Word Signed
X	10000010	SR		LIM	mulhhwu[o][.]	Multiply High Halfword to Word Unsigned
XO	7C000096	SR		B	mulhw[.]	Multiply High Word
XO	7C000016	SR		B	mulhwu[.]	Multiply High Word Unsigned
XO	7C0001D2	SR		64	mulld[o][.]	Multiply Low Doubleword
XO	7C0001D6	SR		B	mullw[o][.]	Multiply Low Word
X	7C0003B8	SR		B	nand[.]	NAND
X	7C0000D0	SR		B	neg[o][.]	Negate
X	1000015C	SR		LIM	nmacchw[o][.]	Negative Multiply Accumulate Cross Halfword to Word Modulo Signed
X	100001DC	SR		LIM	nmacchws[o][.]	Negative Multiply Accumulate Cross Halfword to Word Saturate Signed
X	1000005C	SR		LIM	nmachhw[o][.]	Negative Multiply Accumulate High Halfword to Word Modulo Signed
X	100000DC	SR		LIM	nmachhws[o][.]	Negative Multiply Accumulate High Halfword to Word Saturate Signed

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	1000035C	SR		LIM	nmaclhw [o][.]	Negative Multiply Accumulate Low Halfword to Word Modulo Signed
X	100003DC	SR		LIM	nmaclhws [o][.]	Negative Multiply Accumulate Low Halfword to Word Saturate Signed
X	7C0000F8	SR		B	nor [.]	NOR
X	7C000378	SR		B	or [.]	OR
X	7C000338	SR		B	orc [.]	OR with Complement
X	7C0000F4			B	popcntb	Population Count Bytes
RR	0400----			VLE	se_add	Add Short Form
OIM5	2000----			VLE	se_addi	Add Immediate Short Form
RR	4600----	SR		VLE	se_and [.]	AND Short Form
RR	4500----			VLE	se_andc	AND with Complement Short Form
IM5	2E00----			VLE	se_andi	AND Immediate Short Form
BD8	E800----			VLE	se_b[l]	Branch [and Link]
BD8	E000----			VLE	se_bc	Branch Conditional Short Form
IM5	6000----			VLE	se_bclri	Bit Clear Immediate
C	0006----			VLE	se_bctr	Branch To Count Register [and Link]
IM5	6200----			VLE	se_bgeni	Bit Generate Immediate
C	0004----			VLE	se_blr	Branch To Link Register [and Link]
IM5	2C00----			VLE	se_bmaski	Bit Mask Generate Immediate
IM5	6400----			VLE	se_bseti	Bit Set Immediate
IM5	6600----			VLE	se_btsti	Bit Test Immediate
RR	0C00----			VLE	se_cmp	Compare Word
RR	0E00----			VLE	se_cmph	Compare Halfword Short Form
RR	0F00----			VLE	se_cmphi	Compare Halfword Logical Short Form
IM5	2A00----			VLE	se_cmpi	Compare Immediate Word Short Form
RR	0D00----			VLE	se_cmpl	Compare Logical Word
OIM5	2200----			VLE	se_cmpli	Compare Logical Immediate Word
R	00D0----			VLE	se_extsb	Extend Sign Byte Short Form
R	00F0----			VLE	se_extsh	Extend Sign Halfword Short Form
R	00C0----			VLE	se_extzb	Extend Zero Byte
R	00E0----			VLE	se_extzh	Extend Zero Halfword
C	0000----			VLE	se_illegal	Illegal

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
C	0001----			VLE	se_isync	Instruction Synchronize
SD4	8000----			VLE	se_lbz	Load Byte and Zero Short Form
SD4	A000----			VLE	se_lhz	Load Halfword and Zero Short Form
IM7	4800----			VLE	se_li	Load Immediate Short Form
SD4	C000----			VLE	se_lwz	Load Word and Zero Short Form
RR	0300----			VLE	se_mfar	Move from Alternate Register
R	00A0----			VLE	se_mfctr	Move From Count Register
R	0080----			VLE	se_mflr	Move From Link Register
RR	0100----			VLE	se_mr	Move Register
RR	0200----			VLE	se_mtar	Move To Alternate Register
R	00B0----			VLE	se_mtctr	Move To Count Register
R	0090----			VLE	se_mtr	Move To Link Register
RR	0500----			VLE	se_mullw	Multiply Low Word Short Form
R	0030----			VLE	se_neg	Negate Short Form
R	0020----			VLE	se_not	NOT Short Form
RR	4400----			VLE	se_or	OR Short Form
C	0009----		P	VLE	se_rfc	Return From Critical Interrupt
C	000A----		P	VLE	se_rfdi	Return From Debug Interrupt
C	0008----		P	VLE	se_rfi	Return from Interrupt
C	000B----		P	VLE	se_rfmci	Return From Machine Check Interrupt
C	0002----			VLE	se_sc	System Call
RR	4200----			VLE	se_slw	Shift Left Word
IM5	6C00----			VLE	se_slwi	Shift Left Word Immediate Short Form
RR	4100----	SR		VLE	se_sraw	Shift Right Algebraic Word
IM5	6A00----	SR		VLE	se_srawi	Shift Right Algebraic Immediate
RR	4000----			VLE	se_srw	Shift Right Word
IM5	6800----			VLE	se_srwi	Shift Right Word Immediate Short Form
SD4	9000----			VLE	se_stb	Store Byte Short Form
SD4	B000----			VLE	se_sth	Store Halfword Short Form
SD4	D000----			VLE	se_stw	Store Word Short Form
RR	0600----			VLE	se_sub	Subtract
RR	0700----			VLE	se_subf	Subtract From Short Form

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
OIM5	2400----	SR		VLE	se_subi [.]	Subtract Immediate
X	7C000036	SR		64	sld [.]	Shift Left Doubleword
X	7C000030	SR		B	slw [.]	Shift Left Word
X	7C000634	SR		64	srad [.]	Shift Right Algebraic Doubleword
X	7C000674	SR		64	sradi [.]	Shift Right Algebraic Doubleword Immediate
X	7C000630	SR		B	sraw [.]	Shift Right Algebraic Word
X	7C000670	SR		B	srawi [.]	Shift Right Algebraic Word Immediate
X	7C000436	SR		64	srd [.]	Shift Right Doubleword
X	7C000430	SR		B	srw [.]	Shift Right Word
X	7C0001BE		P	E.PD	stbepx	Store Byte by External Process ID Indexed
X	7C0001EE			B	stbux	Store Byte with Update Indexed
X	7C0001AE			B	stbx	Store Bye Indexed
X	7C0001AD			64	stdcx.	Store Doubleword Conditional Indexed
X	7C00013A		P	E.PD	stdepix	Store Doubleword by External Process ID Indexed
X	7C00016A			64	stdux	Store Doubleword with Update Indexed
X	7C00012A			64	stdx	Store Doubleword Indexed
X	7C0005BE		P	E.PD	stfdepix	Store Floating-Point Double by External Process ID Indexed
X	7C00072C			B	sthbrx	Store Halfword Byte-Reversed Indexed
X	7C00033E		P	E.PD	sthepix	Store Halfword by External Process ID Indexed
X	7C00036E			B	sthux	Store Halfword with Update Indexed
X	7C00032E			B	sthx	Store Halfword Indexed
X	7C0005AA			MA	stswi	Store String Word Immediate
X	7C00052A			MA	stswx	Store String Word Indexed
VX	7C00010E			VEC	stvebx	Store Vector Element Byte Indexed
VX	7C00014E			VEC	stvehx	Store Vector Element Halfword Indexed
X	7C00064E		P	E.PD	stvepix	Store Vector by External Process ID Indexed
X	7C00060E		P	E.PD	stvepixl	Store Vector by External Process ID Indexed LRU
VX	7C00018E			VEC	stviewx	Store Vector Element Word Indexed
VX	7C0001CE			VEC	stvx [!]	Store Vector Indexed [Last]
X	7C00052C			B	stwbrx	Store Word Byte-Reversed Indexed
X	7C00012D			B	stwcx.	Store Word Conditional Indexed

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C00013E		P	E.PD	stwepx	Store Word by External Process ID Indexed
X	7C00016E			B	stwux	Store Word with Update Indexed
X	7C00012E			B	stwx	Store Word Indexed
XO	7C000050	SR		B	subf[o][.]	Subtract From
XO	7C000010	SR		B	subfc[o][.]	Subtract From Carrying
XO	7C000110	SR		B	subfe[o][.]	Subtract From Extended
XO	7C0001D0	SR		B	subfme[o][.]	Subtract From Minus One Extended
XO	7C000190	SR		B	subfze[o][.]	Subtract From Zero Extended
X	7C0004AC			B	sync	Synchronize
X	7C000088			64	td	Trap Doubleword
X	7C000624		P	E	tlbivax	TLB Invalidate Virtual Address Indexed
X	7C000764		P	E	tlbre	TLB Read Entry
X	7C000724		P	E	tlbsx	TLB Search Indexed
X	7C00046C		P	E	tlbsync	TLB Synchronize
X	7C0007A4		P	E	tlbwe	TLB Write Entry
X	7C000008			B	tw	Trap Word
VX	10000180			VEC	vaddcuw	Vector Add Carryout Unsigned Word
VX	1000000A			VEC	vaddfp	Vector Add Floating-Point
VX	10000300			VEC	vaddsbs	Vector Add Signed Byte Saturate
VX	10000340			VEC	vaddshs	Vector Add Signed Halfword Saturate
VX	10000380			VEC	vaddsws	Vector Add Signed Word Saturate
VX	10000000			VEC	vaddubm	Vector Add Unsigned Byte Modulo
VX	10000200			VEC	vaddubs	Vector Add Unsigned Byte Saturate
VX	10000040			VEC	vadduhm	Vector Add Unsigned Halfword Modulo
VX	10000240			VEC	vadduhs	Vector Add Unsigned Halfword Saturate
VX	10000080			VEC	vadduwm	Vector Add Unsigned Word Modulo
VX	10000280			VEC	vadduws	Vector Add Unsigned Word Saturate
VX	10000404			VEC	vand	Vector AND
VX	10000444			VEC	vandc	Vector AND with Complement
VX	10000502			VEC	vavgsb	Vector Average Signed Byte
VX	10000542			VEC	vavgsh	Vector Average Signed Halfword
VX	10000582			VEC	vavgsw	Vector Average Signed Word

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	10000402			VEC	vavgub	Vector Average Unsigned Byte
VX	10000442			VEC	vavguh	Vector Average Unsigned Halfword
VX	10000482			VEC	vavguw	Vector Average Unsigned Word
VX	100003CA			VEC	vcfpsxws	Vector Convert from Single-Precision to Signed Fixed-Point Word Saturate
VX	1000038A			VEC	vcfpuwxs	Vector Convert from Single-Precision to Unsigned Fixed-Point Word Saturate
VX	100003C6			VEC	vcmpbfp[.]	Vector Compare Bounds Single-Precision
VX	100000C6			VEC	vcmpeqfp[.]	Vector Compare Equal To Single-Precision
VX	10000006			VEC	vcmpequb[.]	Vector Compare Equal To Unsigned Byte
VX	10000046			VEC	vcmpequh[.]	Vector Compare Equal To Unsigned Halfword
VX	10000086			VEC	vcmpequw[.]	Vector Compare Equal To Unsigned Word
VX	100001C6			VEC	vcmpgef[.]	Vector Compare Greater Than or Equal To Single-Precision
VX	100002C6			VEC	vcmpgtfp[.]	Vector Compare Greater Than Single-Precision
VX	10000306			VEC	vcmpgtsb[.]	Vector Compare Greater Than Signed Byte
VX	10000346			VEC	vcmpgtsh[.]	Vector Compare Greater Than Signed Halfword
VX	10000386			VEC	vcmpgtsw[.]	Vector Compare Greater Than Signed Word
VX	10000206			VEC	vcmpgtub[.]	Vector Compare Greater Than Unsigned Byte
VX	10000246			VEC	vcmpgtuh[.]	Vector Compare Greater Than Unsigned Halfword
VX	10000286			VEC	vcmpgtuw[.]	Vector Compare Greater Than Unsigned Word
VX	1000034A			VEC	vcsxwfp	Vector Convert from Signed Fixed-Point Word to Single-Precision
VX	1000030A			VEC	vcuxwfp	Vector Convert from Unsigned Fixed-Point Word to Single-Precision
VX	1000018A			VEC	vexptefp	Vector 2 Raised to the Exponent Estimate Floating-Point
VX	100001CA			VEC	vlogefp	Vector Log Base 2 Estimate Floating-Point
VA	1000002E			VEC	vmaddfp	Vector Multiply-Add Single-Precision
VX	1000040A			VEC	vmaxfp	Vector Maximum Single-Precision
VX	10000102			VEC	vmaxsb	Vector Maximum Signed Byte
VX	10000142			VEC	vmaxsh	Vector Maximum Signed Halfword
VX	10000182			VEC	vmaxsw	Vector Maximum Signed Word
VX	10000002			VEC	vmaxub	Vector Maximum Unsigned Byte
VX	10000042			VEC	vmaxuh	Vector Maximum Unsigned Halfword

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	1000082			VEC	vmaxuw	Vector Maximum Unsigned Word
VA	1000020			VEC	vmhaddshs	Vector Multiply-High-Add Signed Halfword Saturate
VA	1000021			VEC	vmhraddshs	Vector Multiply-High-Round-Add Signed Halfword Saturate
VX	1000044A			VEC	vminfp	Vector Minimum Single-Precision
VX	10000302			VEC	vminsb	Vector Minimum Signed Byte
VX	10000342			VEC	vminsh	Vector Minimum Signed Halfword
VX	10000382			VEC	vminsw	Vector Minimum Signed Word
VX	10000202			VEC	vminub	Vector Minimum Unsigned Byte
VX	10000242			VEC	vminuh	Vector Minimum Unsigned Halfword
VX	10000282			VEC	vminuw	Vector Minimum Unsigned Word
VA	10000022			VEC	vmladduhm	Vector Multiply-Low-Add Unsigned Halfword Modulo
VX	1000000C			VEC	vmrghb	Vector Merge High Byte
VX	1000004C			VEC	vmrghh	Vector Merge High Halfword
VX	1000008C			VEC	vmrghw	Vector Merge High Word
VX	1000010C			VEC	vmrglb	Vector Merge Low Byte
VX	1000014C			VEC	vmrglh	Vector Merge Low Halfword
VX	1000018C			VEC	vmrglw	Vector Merge Low Word
VA	10000025			VEC	vmsummbm	Vector Multiply-Sum Mixed Byte Modulo
VA	10000028			VEC	vmsumshm	Vector Multiply-Sum Signed Halfword Modulo
VA	10000029			VEC	vmsumshs	Vector Multiply-Sum Signed Halfword Saturate
VA	10000024			VEC	vmsumubm	Vector Multiply-Sum Unsigned Byte Modulo
VA	10000026			VEC	vmsumuhm	Vector Multiply-Sum Unsigned Halfword Modulo
VA	10000027			VEC	vmsumuhs	Vector Multiply-Sum Unsigned Halfword Saturate
VX	10000308			VEC	vmulesb	Vector Multiply Even Signed Byte
VX	10000348			VEC	vmulesh	Vector Multiply Even Signed Halfword
VX	10000208			VEC	vmuleub	Vector Multiply Even Unsigned Byte
VX	10000248			VEC	vmuleuh	Vector Multiply Even Unsigned Halfword
VX	10000108			VEC	vmulosb	Vector Multiply Odd Signed Byte
VX	10000148			VEC	vmulosh	Vector Multiply Odd Signed Halfword
VX	10000008			VEC	vmuloub	Vector Multiply Odd Unsigned Byte
VX	10000048			VEC	vmulouh	Vector Multiply Odd Unsigned Halfword

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VA	100002F			VEC	vnmsubfp	Vector Negative Multiply-Subtract Single-Precision
VX	10000504			VEC	vnor	Vector NOR
VX	10000484			VEC	vor	Vector OR
VA	1000002B			VEC	vperm	Vector Permute
VX	1000030E			VEC	vpkpx	Vector Pack Pixel
VX	1000018E			VEC	vpkshss	Vector Pack Signed Halfword Signed Saturate
VX	1000010E			VEC	vpkshus	Vector Pack Signed Halfword Unsigned Saturate
VX	100001CE			VEC	vpkswss	Vector Pack Signed Word Signed Saturate
VX	1000014E			VEC	vpkswus	Vector Pack Signed Word Unsigned Saturate
VX	1000000E			VEC	vpkuhum	Vector Pack Unsigned Halfword Unsigned Modulo
VX	1000008E			VEC	vpkuhus	Vector Pack Unsigned Halfword Unsigned Saturate
VX	1000004E			VEC	vpkuwum	Vector Pack Unsigned Word Unsigned Modulo
VX	100000CE			VEC	vpkuwus	Vector Pack Unsigned Word Unsigned Saturate
VX	1000010A			VEC	vrffp	Vector Reciprocal Estimate Single-Precision
VX	100002CA			VEC	vrfin	Vector Round to Single-Precision Integer toward Minus Infinity
VX	1000020A			VEC	vrfin	Vector Round to Single-Precision Integer Nearest
VX	1000028A			VEC	vrrip	Vector Round to Single-Precision Integer toward Positive Infinity
VX	1000024A			VEC	vrifz	Vector Round to Single-Precision Integer toward Zero
VX	10000004			VEC	vrlb	Vector Rotate Left Byte
VX	10000044			VEC	vrlh	Vector Rotate Left Halfword
VX	10000084			VEC	vrlw	Vector Rotate Left Word
VX	1000014A			VEC	vrsqrtefp	Vector Reciprocal Square Root Estimate Single-Precision
VA	1000002A			VEC	vsel	Vector Select
VX	100001C4			VEC	vsl	Vector Shift Left
VX	10000104			VEC	vslb	Vector Shift Left Byte
VA	1000002C			VEC	vsldoi	Vector Shift Left Double by Octet Immediate
VX	10000144			VEC	vslh	Vector Shift Left Halfword
VX	1000040C			VEC	vslo	Vector Shift Left by Octet
VX	10000184			VEC	vslw	Vector Shift Left Word
VX	1000020C			VEC	vspltb	Vector Splat Byte

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	1000024C			VEC	vsplth	Vector Splat Halfword
VX	1000030C			VEC	vspltisb	Vector Splat Immediate Signed Byte
VX	1000034C			VEC	vspltish	Vector Splat Immediate Signed Halfword
VX	1000038C			VEC	vspltisw	Vector Splat Immediate Signed Word
VX	1000028C			VEC	vspltw	Vector Splat Word
VX	100002C4			VEC	vsr	Vector Shift Right
VX	10000304			VEC	vsrab	Vector Shift Right Algebraic Word
VX	10000344			VEC	vsrah	Vector Shift Right Algebraic Word
VX	10000384			VEC	vsraw	Vector Shift Right Algebraic Word
VX	10000204			VEC	vsrb	Vector Shift Right Byte
VX	10000244			VEC	vsrh	Vector Shift Right Halfword
VX	1000044C			VEC	vsro	Vector Shift Right by Octet
VX	10000284			VEC	vsrw	Vector Shift Right Word
VX	10000580			VEC	vsubcuw	Vector Subtract and Write Carry-Out Unsigned Word
VX	1000004A			VEC	vsubfp	Vector Subtract Single-Precision
VX	10000700			VEC	vsubsbbs	Vector Subtract Signed Byte Saturate
VX	10000740			VEC	vsubshs	Vector Subtract Signed Halfword Saturate
VX	10000780			VEC	vsubsws	Vector Subtract Signed Word Saturate
VX	10000400			VEC	vsububm	Vector Subtract Unsigned Byte Modulo
VX	10000600			VEC	vsububs	Vector Subtract Unsigned Byte Saturate
VX	10000440			VEC	vsubuhm	Vector Subtract Unsigned Byte Modulo
VX	10000640			VEC	vsubuhs	Vector Subtract Unsigned Halfword Saturate
VX	10000480			VEC	vsubuwm	Vector Subtract Unsigned Word Modulo
VX	10000680			VEC	vsubuws	Vector Subtract Unsigned Word Saturate
VX	10000688			VEC	vsum2sws	Vector Sum across Half Signed Word Saturate
VX	10000708			VEC	vsum4sbs	Vector Sum across Quarter Signed Byte Saturate
VX	10000648			VEC	vsum4shs	Vector Sum across Quarter Signed Halfword Saturate
VX	10000608			VEC	vsum4ubs	Vector Sum across Quarter Unsigned Byte Saturate
VX	10000788			VEC	vsumsws	Vector Sum across Signed Word Saturate
VX	1000034E			VEC	vupkhp	Vector Unpack High Pixel
VX	1000020E			VEC	vupkhsb	Vector Unpack High Signed Byte
VX	1000024E			VEC	vupkhsh	Vector Unpack High Signed Halfword

Table B-2. VLE Instruction Set Sorted by Mnemonic (continued)

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	100003CE			VEC	vupklpx	Vector Unpack Low Pixel
VX	1000028E			VEC	vupklsb	Vector Unpack Low Signed Byte
VX	100002CE			VEC	vupklsh	Vector Unpack Low Signed Halfword
VX	100004C4			VEC	vxor	Vector XOR
X	7C00007C			WT	wait	Wait
X	7C000106		P	E	wrtee	Write MSR External Enable
X	7C000146		P	E	wrteei	Write MSR External Enable Immediate
D	7C000278	SR		B	xor[.]	XOR

¹ For 16-bit instructions, this column represents the 16-bit hexadecimal instruction encoding with the opcode and extended opcode in the corresponding fields in the instruction, and with 0s in bit positions that are not opcode bits; dashes are used following the opcode to indicate the form is a 16-bit instruction. For 32-bit instructions, this column represents the 32-bit hexadecimal instruction encoding with the opcode and extended opcode in the corresponding fields in the instruction, and with 0s in bit positions that are not opcode bits.

² See the key to the mode dependency and privilege columns in [Table B-1](#).

B.2 VLE Instruction Set Sorted by Opcode

[Table B-3](#) lists the instructions available in VLE mode in the PowerPC Architecture, in order by opcode. Opcodes that are not defined below are treated as illegal by VLE.

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
C	0000----			VLE	se_illegal	Illegal
C	0001----			VLE	se_isync	Instruction Synchronize
C	0002----			VLE	se_sc	System Call
C	0004----			VLE	se_blr	Branch To Link Register [and Link]
C	0006----			VLE	se_bctr	Branch To Count Register [and Link]
C	0008----		P	VLE	se_rfi	Return from Interrupt
C	0009----		P	VLE	se_rfci	Return From Critical Interrupt
C	000A----		P	VLE	se_rfdi	Return From Debug Interrupt
C	000B----		P	VLE	se_rfmci	Return From Machine Check Interrupt
R	0020----			VLE	se_not	NOT Short Form
R	0030----			VLE	se_neg	Negate Short Form
R	0080----			VLE	se_mflr	Move From Link Register
R	0090----			VLE	se_mtlr	Move To Link Register

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
R	00A0----			VLE	se_mfctr	Move From Count Register
R	00B0----			VLE	se_mtctr	Move To Count Register
R	00C0----			VLE	se_extzb	Extend Zero Byte
R	00D0----			VLE	se_extsb	Extend Sign Byte Short Form
R	00E0----			VLE	se_extzh	Extend Zero Halfword
R	00F0----			VLE	se_extsh	Extend Sign Halfword Short Form
RR	0100----			VLE	se_mr	Move Register
RR	0200----			VLE	se_mtar	Move To Alternate Register
RR	0300----			VLE	se_mfar	Move from Alternate Register
RR	0400----			VLE	se_add	Add Short Form
RR	0500----			VLE	se_mullw	Multiply Low Word Short Form
RR	0600----			VLE	se_sub	Subtract
RR	0700----			VLE	se_subf	Subtract From Short Form
RR	0C00----			VLE	se_cmp	Compare Word
RR	0D00----			VLE	se_cmpl	Compare Logical Word
RR	0E00----			VLE	se_cmph	Compare Halfword Short Form
RR	0F00----			VLE	se_cmphi	Compare Halfword Logical Short Form
VX	10000000			VEC	vaddubm	Vector Add Unsigned Byte Modulo
VX	10000002			VEC	vmaxub	Vector Maximum Unsigned Byte
VX	10000004			VEC	vrlb	Vector Rotate Left Byte
VX	10000006			VEC	vcmpesub[.]	Vector Compare Equal To Unsigned Byte
VX	10000008			VEC	vmuloub	Vector Multiply Odd Unsigned Byte
VX	1000000A			VEC	vaddfp	Vector Add Floating-Point
VX	1000000C			VEC	vmrghb	Vector Merge High Byte
VX	1000000E			VEC	vpkuhum	Vector Pack Unsigned Halfword Unsigned Modulo
X	10000010	SR		LIM	mulhhu[o][.]	Multiply High Halfword to Word Unsigned
X	10000018	SR		LIM	machhu[o][.]	Multiply Accumulate High Halfword to Word Modulo Unsigned
VA	10000020			VEC	vmhaddshs	Vector Multiply-High-Add Signed Halfword Saturate
VA	10000021			VEC	vmhraddshs	Vector Multiply-High-Round-Add Signed Halfword Saturate
VA	10000022			VEC	vmladduhm	Vector Multiply-Low-Add Unsigned Halfword Modulo
VA	10000024			VEC	vmsumubm	Vector Multiply-Sum Unsigned Byte Modulo

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VA	1000025			VEC	vmsummbm	Vector Multiply-Sum Mixed Byte Modulo
VA	1000026			VEC	vmsumuhm	Vector Multiply-Sum Unsigned Halfword Modulo
VA	1000027			VEC	vmsumuhs	Vector Multiply-Sum Unsigned Halfword Saturate
VA	1000028			VEC	vmsumshm	Vector Multiply-Sum Signed Halfword Modulo
VA	1000029			VEC	vmsumshs	Vector Multiply-Sum Signed Halfword Saturate
VA	100002A			VEC	vsel	Vector Select
VA	100002B			VEC	vperm	Vector Permute
VA	100002C			VEC	vsldoi	Vector Shift Left Double by Octet Immediate
VA	100002E			VEC	vmaddfp	Vector Multiply-Add Single-Precision
VA	100002F			VEC	vnmsubfp	Vector Negative Multiply-Subtract Single-Precision
VX	1000040			VEC	vadduhm	Vector Add Unsigned Halfword Modulo
VX	1000042			VEC	vmaxuh	Vector Maximum Unsigned Halfword
VX	1000044			VEC	vrlh	Vector Rotate Left Halfword
VX	1000046			VEC	vcmpquh[.]	Vector Compare Equal To Unsigned Halfword
VX	1000048			VEC	vmulouh	Vector Multiply Odd Unsigned Halfword
VX	100004A			VEC	vsubfp	Vector Subtract Single-Precision
VX	100004C			VEC	vmrghh	Vector Merge High Halfword
VX	100004E			VEC	vpkuwum	Vector Pack Unsigned Word Unsigned Modulo
X	1000050	SR		LIM	mulhhw[o][.]	Multiply High Halfword to Word Signed
X	1000058	SR		LIM	machhw[o][.]	Multiply Accumulate High Halfword to Word Modulo Signed
X	100005C	SR		LIM	nmachhw[o][.]	Negative Multiply Accumulate High Halfword to Word Modulo Signed
VX	1000080			VEC	vadduwm	Vector Add Unsigned Word Modulo
VX	1000082			VEC	vmaxuw	Vector Maximum Unsigned Word
VX	1000084			VEC	vrlw	Vector Rotate Left Word
VX	1000086			VEC	vcmpquw[.]	Vector Compare Equal To Unsigned Word
VX	100008C			VEC	vmrghw	Vector Merge High Word
VX	100008E			VEC	vpkuhus	Vector Pack Unsigned Halfword Unsigned Saturate
X	1000098	SR		LIM	machwsu[o][.]	Multiply Accumulate High Halfword to Word Saturate Unsigned
VX	10000C6			VEC	vcmpqfp[.]	Vector Compare Equal To Single-Precision
VX	10000CE			VEC	vpkuwus	Vector Pack Unsigned Word Unsigned Saturate

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	10000D8	SR		LIM	machws[o][.]	Multiply Accumulate High Halfword to Word Saturate Signed
X	10000DC	SR		LIM	nmachws[o][.]	Negative Multiply Accumulate High Halfword to Word Saturate Signed
VX	10000102			VEC	vmaxsb	Vector Maximum Signed Byte
VX	10000104			VEC	vslb	Vector Shift Left Byte
VX	10000108			VEC	vmulosb	Vector Multiply Odd Signed Byte
VX	1000010A			VEC	vrefp	Vector Reciprocal Estimate Single-Precision
VX	1000010C			VEC	vmrglb	Vector Merge Low Byte
VX	1000010E			VEC	vpkshus	Vector Pack Signed Halfword Unsigned Saturate
X	10000110	SR		LIM	mulchwu[o][.]	Multiply Cross Halfword to Word Unsigned
X	10000118	SR		LIM	macchwu[o][.]	Multiply Accumulate Cross Halfword to Word Modulo Unsigned
VX	10000142			VEC	vmaxsh	Vector Maximum Signed Halfword
VX	10000144			VEC	vslh	Vector Shift Left Halfword
VX	10000148			VEC	vmulosh	Vector Multiply Odd Signed Halfword
VX	1000014A			VEC	vsqrtefp	Vector Reciprocal Square Root Estimate Single-Precision
VX	1000014C			VEC	vmrglh	Vector Merge Low Halfword
VX	1000014E			VEC	vpkswus	Vector Pack Signed Word Unsigned Saturate
X	10000150	SR		LIM	mulchw[o][.]	Multiply Cross Halfword to Word Signed
X	10000158	SR		LIM	macchw[o][.]	Multiply Accumulate Cross Halfword to Word Modulo Signed
X	1000015C	SR		LIM	nmacchw[o][.]	Negative Multiply Accumulate Cross Halfword to Word Modulo Signed
VX	10000180			VEC	vaddcuw	Vector Add Carryout Unsigned Word
VX	10000182			VEC	vmaxsw	Vector Maximum Signed Word
VX	10000184			VEC	vslw	Vector Shift Left Word
VX	1000018A			VEC	vexptefp	Vector 2 Raised to the Exponent Estimate Floating-Point
VX	1000018C			VEC	vmrglw	Vector Merge Low Word
VX	1000018E			VEC	vpkshss	Vector Pack Signed Halfword Signed Saturate
X	10000198	SR		LIM	macchwsu[o][.]	Multiply Accumulate Cross Halfword to Word Saturate Unsigned
VX	100001C4			VEC	vsl	Vector Shift Left

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	10001C6			VEC	vcmpgefp [.]	Vector Compare Greater Than or Equal To Single-Precision
VX	10001CA			VEC	vlogefp	Vector Log Base 2 Estimate Floating-Point
VX	10001CE			VEC	vpkswss	Vector Pack Signed Word Signed Saturate
X	10001D8	SR		LIM	macchws [o][.]	Multiply Accumulate Cross Halfword to Word Saturate Signed
X	10001DC	SR		LIM	nmacchws [o][.]	Negative Multiply Accumulate Cross Halfword to Word Saturate Signed
EVX	1000200			SP	evaddw	Vector Add Word
VX	1000200			VEC	vaddubs	Vector Add Unsigned Byte Saturate
EVX	1000202			SP	evaddiw	Vector Add Immediate Word
VX	1000202			VEC	vminub	Vector Minimum Unsigned Byte
EVX	1000204			SP	evsubfw	Vector Subtract from Word
VX	1000204			VEC	vsrb	Vector Shift Right Byte
EVX	1000206			SP	evsubifw	Vector Subtract Immediate from Word
VX	1000206			VEC	vcmpgtub [.]	Vector Compare Greater Than Unsigned Byte
EVX	1000208			SP	evabs	Vector Absolute Value
VX	1000208			VEC	vmuleub	Vector Multiply Even Unsigned Byte
EVX	1000209			SP	evneg	Vector Negate
EVX	100020A			SP	evextsb	Vector Extend Sign Byte
VX	100020A			VEC	vrfin	Vector Round to Single-Precision Integer Nearest
EVX	100020B			SP	evextsh	Vector Extend Sign Halfword
EVX	100020C			SP	evrndw	Vector Round Word
VX	100020C			VEC	vspltb	Vector Splat Byte
EVX	100020D			SP	evcntlzw	Vector Count Leading Zeros Bits Word
EVX	100020E			SP	evcntlsw	Vector Count Leading Sign Bits Word
VX	100020E			VEC	vupkhsb	Vector Unpack High Signed Byte
EVX	100020F			SP	brinc	Bit Reverse Increment
EVX	1000211			SP	evand	Vector AND
EVX	1000212			SP	evandc	Vector AND with Complement
EVX	1000216			SP	evxor	Vector XOR
EVX	1000217			SP	evor	Vector OR
EVX	1000218			SP	evnor	Vector NOR

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000219			SP	eveqv	Vector Equivalent
EVX	100021B			SP	evorc	Vector OR with Complement
EVX	100021E			SP	evnand	Vector NAND
EVX	1000220			SP	evsrwu	Vector Shift Right Word Unsigned
EVX	1000221			SP	evsrws	Vector Shift Right Word Signed
EVX	1000222			SP	evsrwiu	Vector Shift Right Word Immediate Unsigned
EVX	1000223			SP	evsrwis	Vector Shift Right Word Immediate Signed
EVX	1000224			SP	evslw	Vector Shift Left Word
EVX	1000226			SP	evslwi	Vector Shift Left Word Immediate
EVX	1000228			SP	evrlw	Vector Rotate Left Word
EVX	1000229			SP	evsplati	Vector Splat Immediate
EVX	100022A			SP	evrlwi	Vector Rotate Left Word Immediate
EVX	100022B			SP	evsplatfi	Vector Splat Fractional Immediate
EVX	100022C			SP	evmergehi	Vector Merge High
EVX	100022D			SP	evmergelo	Vector Merge Low
EVX	100022E			SP	evmergehilo	Vector Merge High/Low
EVX	100022F			SP	evmergelohi	Vector Merge Low/High
EVX	1000230			SP	evcmpgtu	Vector Compare Greater Than Unsigned
EVX	1000231			SP	evcmpgts	Vector Compare Greater Than Signed
EVX	1000232			SP	evcmpltu	Vector Compare Less Than Unsigned
EVX	1000233			SP	evcmplts	Vector Compare Less Than Signed
EVX	1000234			SP	evcmpeq	Vector Compare Equal
VX	1000240			VEC	vadduhs	Vector Add Unsigned Halfword Saturate
VX	1000242			VEC	vminuh	Vector Minimum Unsigned Halfword
VX	1000244			VEC	vsrh	Vector Shift Right Halfword
VX	1000246			VEC	vcmpgtuh[.]	Vector Compare Greater Than Unsigned Halfword
VX	1000248			VEC	vmuleuh	Vector Multiply Even Unsigned Halfword
VX	100024A			VEC	vrfiz	Vector Round to Single-Precision Integer toward Zero
VX	100024C			VEC	vsplth	Vector Splat Halfword
VX	100024E			VEC	vupksh	Vector Unpack High Signed Halfword
EVSEL	1000278			SP	evsel	Vector Select
EVX	1000280			SP.FV	evfsadd	Vector Floating-Point Single-Precision Add

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	1000280			VEC	vadduws	Vector Add Unsigned Word Saturate
EVX	1000281			SP.FV	evfssub	Vector Floating-Point Single-Precision Subtract
VX	1000282			VEC	vminuw	Vector Minimum Unsigned Word
EVX	1000284			SP.FV	evfsabs	Vector Floating-Point Single-Precision Absolute Value
VX	1000284			VEC	vsrw	Vector Shift Right Word
EVX	1000285			SP.FV	evfsnabs	Vector Floating-Point Single-Precision Negative Absolute Value
EVX	1000286			SP.FV	evfsneg	Vector Floating-Point Single-Precision Negate
VX	1000286			VEC	vcmpgtuw[.]	Vector Compare Greater Than Unsigned Word
EVX	1000288			SP.FV	evfsmul	Vector Floating-Point Single-Precision Multiply
EVX	1000289			SP.FV	evfsdiv	Vector Floating-Point Single-Precision Divide
VX	100028A			VEC	vrrip	Vector Round to Single-Precision Integer toward Positive Infinity
EVX	100028C			SP.FV	evfscmpgt	Vector Floating-Point Single-Precision Compare Greater Than
VX	100028C			VEC	vspltw	Vector Splat Word
EVX	100028D			SP.FV	evfscmplt	Vector Floating-Point Single-Precision Compare Less Than
EVX	100028E			SP.FV	evfscmpeq	Vector Floating-Point Single-Precision Compare Equal
VX	100028E			VEC	vupklbs	Vector Unpack Low Signed Byte
EVX	1000290			SP.FV	evfscfui	Vector Convert Floating-Point Single-Precision from Unsigned Integer
EVX	1000291			SP.FV	evfscfsi	Vector Convert Floating-Point Single-Precision from Signed Integer
EVX	1000292			SP.FV	evfscfuf	Vector Convert Floating-Point Single-Precision from Unsigned Fraction
EVX	1000293			SP.FV	evfscfsf	Vector Convert Floating-Point Single-Precision from Signed Fraction
EVX	1000294			SP.FV	evfsctui	Vector Convert Floating-Point Single-Precision to Unsigned Integer
EVX	1000295			SP.FV	evfsctsi	Vector Convert Floating-Point Single-Precision to Signed Integer
EVX	1000296			SP.FV	evfsctuf	Vector Convert Floating-Point Single-Precision to Unsigned Fraction
EVX	1000297			SP.FV	evfsctsf	Vector Convert Floating-Point Single-Precision to Signed Fraction

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000298			SP.FV	evfsctuiz	Vector Convert Floating-Point Single-Precision to Unsigned Integer with Round Towards Zero
EVX	100029A			SP.FV	evfsctsiz	Vector Convert Floating-Point Single-Precision to Signed Integer with Round Towards Zero
EVX	100029C			SP.FV	evfststgt	Vector Floating-Point Single-Precision Test Greater Than
EVX	100029D			SP.FV	evfststlt	Vector Floating-Point Single-Precision Test Less Than
EVX	100029E			SP.FV	evfststeq	Vector Floating-Point Single-Precision Test Equal
VX	10002C4			VEC	vsr	Vector Shift Right
VX	10002C6			VEC	vcmpgtfp[.]	Vector Compare Greater Than Single-Precision
VX	10002CA			VEC	vrfim	Vector Round to Single-Precision Integer toward Minus Infinity
VX	10002CE			VEC	vupklsh	Vector Unpack Low Signed Halfword
EVX	10002CF			SP.FD	efscfd	Floating-Point Single-Precision Convert from Double-Precision
EVX	10002E0			SP.FD	efdadd	Floating-Point Double-Precision Add
EVX	10002E0			SP.FS	efsadd	Floating-Point Single-Precision Add
EVX	10002E1			SP.FD	efdsab	Floating-Point Double-Precision Subtract
EVX	10002E1			SP.FS	efssub	Floating-Point Single-Precision Subtract
EVX	10002E2			SP.FD	efdcfuid	Convert Floating-Point Double-Precision from Unsigned Integer Doubleword
EVX	10002E2			SP.FS	efscfuid	Convert Floating-Point Single-Precision from Unsigned Integer Doubleword
EVX	10002E3			SP.FD	efdcfsid	Convert Floating-Point Double-Precision from Signed Integer Doubleword
EVX	10002E3			SP.FS	efscfsid	Convert Floating-Point Single-Precision from Signed Integer Doubleword
EVX	10002E4			SP.FD	efdabs	Floating-Point Double-Precision Absolute Value
EVX	10002E4			SP.FS	efsabs	Floating-Point Single-Precision Absolute Value
EVX	10002E5			SP.FD	efdnabs	Floating-Point Double-Precision Negative Absolute Value
EVX	10002E5			SP.FS	efsnabs	Floating-Point Single-Precision Negative Absolute Value
EVX	10002E6			SP.FD	efdneg	Floating-Point Double-Precision Negate
EVX	10002E6			SP.FS	efsneg	Floating-Point Single-Precision Negate
EVX	10002E8			SP.FD	efdmul	Floating-Point Double-Precision Multiply
EVX	10002E8			SP.FS	efsmul	Floating-Point Single-Precision Multiply
EVX	10002E9			SP.FD	efddiv	Floating-Point Double-Precision Divide

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100002E9			SP.FS	efsddiv	Floating-Point Single-Precision Divide
EVX	100002EA			SP.FD	efdctuidz	Convert Floating-Point Double-Precision to Unsigned Integer Doubleword with Round Towards Zero
EVX	100002EA			SP.FS	efscuidz	Convert Floating-Point Single-Precision to Unsigned Integer Doubleword with Round Towards Zero
EVX	100002EB			SP.FD	efdctsidz	Convert Floating-Point Double-Precision to Signed Integer Doubleword with Round Towards Zero
EVX	100002EB			SP.FS	efscsidz	Convert Floating-Point Single-Precision to Signed Integer Doubleword with Round Towards Zero
EVX	100002EC			SP.FD	efdcmpgt	Floating-Point Double-Precision Compare Greater Than
EVX	100002EC			SP.FS	efscmpgt	Floating-Point Single-Precision Compare Greater Than
EVX	100002ED			SP.FD	efdcmlt	Floating-Point Double-Precision Compare Less Than
EVX	100002ED			SP.FS	efscmlt	Floating-Point Single-Precision Compare Less Than
EVX	100002EE			SP.FD	efdcmpaq	Floating-Point Double-Precision Compare Equal
EVX	100002EE			SP.FS	efscmpaq	Floating-Point Single-Precision Compare Equal
EVX	100002EF			SP.FD	efdcfs	Floating-Point Double-Precision Convert from Single-Precision
EVX	100002F0			SP.FD	efdcfui	Convert Floating-Point Double-Precision from Unsigned Integer
EVX	100002F0			SP.FS	efscfui	Convert Floating-Point Single-Precision from Unsigned Integer
EVX	100002F1			SP.FD	efdcfsi	Convert Floating-Point Double-Precision from Signed Integer
EVX	100002F1			SP.FS	efscfsi	Convert Floating-Point Single-Precision from Signed Integer
EVX	100002F2			SP.FD	efdcfuf	Convert Floating-Point Double-Precision from Unsigned Fraction
EVX	100002F2			SP.FS	efscfuf	Convert Floating-Point Single-Precision from Unsigned Fraction
EVX	100002F3			SP.FD	efdcfsf	Convert Floating-Point Double-Precision from Signed Fraction
EVX	100002F3			SP.FS	efscfsf	Convert Floating-Point Single-Precision from Signed Fraction
EVX	100002F4			SP.FD	efdctui	Convert Floating-Point Double-Precision to Unsigned Integer
EVX	100002F4			SP.FS	efscctui	Convert Floating-Point Single-Precision to Unsigned Integer

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100002F5			SP.FD	efdctsi	Convert Floating-Point Double-Precision to Signed Integer
EVX	100002F5			SP.FS	efsctsi	Convert Floating-Point Single-Precision to Signed Integer
EVX	100002F6			SP.FD	efdctuf	Convert Floating-Point Double-Precision to Unsigned Fraction
EVX	100002F6			SP.FS	efsctuf	Convert Floating-Point Single-Precision to Unsigned Fraction
EVX	100002F7			SP.FD	efdctsf	Convert Floating-Point Double-Precision to Signed Fraction
EVX	100002F7			SP.FS	efsctsf	Convert Floating-Point Single-Precision to Signed Fraction
EVX	100002F8			SP.FD	efdctuiz	Convert Floating-Point Double-Precision to Unsigned Integer with Round Towards Zero
EVX	100002F8			SP.FS	efsctuiz	Convert Floating-Point Single-Precision to Unsigned Integer with Round Towards Zero
EVX	100002FA			SP.FD	efdctsiz	Convert Floating-Point Double-Precision to Signed Integer with Round Towards Zero
EVX	100002FA			SP.FS	efsctsiz	Convert Floating-Point Single-Precision to Signed Integer with Round Towards Zero
EVX	100002FC			SP.FD	efdttgt	Floating-Point Double-Precision Test Greater Than
EVX	100002FC			SP.FS	efsttgt	Floating-Point Single-Precision Test Greater Than
EVX	100002FD			SP.FD	efdttl	Floating-Point Double-Precision Test Less Than
EVX	100002FD			SP.FS	efsttl	Floating-Point Single-Precision Test Less Than
EVX	100002FE			SP.FD	efdteq	Floating-Point Double-Precision Test Equal
EVX	100002FE			SP.FS	efsteq	Floating-Point Single-Precision Test Equal
EVX	10000300			SP	evlddx	Vector Load Doubleword into Doubleword Indexed
VX	10000300			VEC	vaddsbs	Vector Add Signed Byte Saturate
EVX	10000301			SP	evldd	Vector Load Doubleword into Doubleword
EVX	10000302			SP	evldwx	Vector Load Doubleword into 2 Words Indexed
VX	10000302			VEC	vminsb	Vector Minimum Signed Byte
EVX	10000303			SP	evldw	Vector Load Doubleword into 2 Words
EVX	10000304			SP	evldhx	Vector Load Doubleword into 4 Halfwords Indexed
VX	10000304			VEC	vsrab	Vector Shift Right Algebraic Word
EVX	10000305			SP	evldh	Vector Load Doubleword into 4 Halfwords
VX	10000306			VEC	vcmpgtsb[.]	Vector Compare Greater Than Signed Byte

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000308			SP	evlhhesplatx	Vector Load Halfword into Halfwords Even and Splat Indexed
VX	1000308			VEC	vmulesb	Vector Multiply Even Signed Byte
EVX	1000309			SP	evlhhesplat	Vector Load Halfword into Halfwords Even and Splat
VX	100030A			VEC	vcuxwfp	Vector Convert from Unsigned Fixed-Point Word to Single-Precision
EVX	100030C			SP	evlhhusplatx	Vector Load Halfword into Halfwords Odd Unsigned and Splat Indexed
VX	100030C			VEC	vsplitisb	Vector Splat Immediate Signed Byte
EVX	100030D			SP	evlhhusplat	Vector Load Halfword into Halfwords Odd Unsigned and Splat
EVX	100030E			SP	evlhhosplatx	Vector Load Halfword into Halfwords Odd Signed and Splat Indexed
VX	100030E			VEC	vpkpx	Vector Pack Pixel
EVX	100030F			SP	evlhhosplat	Vector Load Halfword into Halfwords Odd and Splat
EVX	1000310			SP	evlwhex	Vector Load Word into Two Halfwords Even Indexed
EVX	1000311			SP	evlwhe	Vector Load Word into Two Halfwords Even
EVX	1000314			SP	evlwhoux	Vector Load Word into Two Halfwords Odd Unsigned Indexed
EVX	1000315			SP	evlwhou	Vector Load Word into Two Halfwords Odd Unsigned
EVX	1000316			SP	evlwhosx	Vector Load Word into Two Halfwords Odd Signed Indexed
EVX	1000317			SP	evlwhos	Vector Load Word into Two Halfwords Odd Signed
EVX	1000318			SP	evlwwsplatx	Vector Load Word into Word and Splat Indexed
X	1000318	SR		LIM	machwu[o][.]	Multiply Accumulate Low Halfword to Word Modulo Unsigned
EVX	1000319			SP	evlwwsplat	Vector Load Word into Word and Splat
EVX	100031C			SP	evlwhsplatx	Vector Load Word into Two Halfwords and Splat Indexed
EVX	100031D			SP	evlwhsplat	Vector Load Word into Two Halfwords and Splat
EVX	1000320			SP	evstddx	Vector Store Doubleword of Doubleword Indexed
EVX	1000321			SP	evstdd	Vector Store Doubleword of Doubleword
EVX	1000322			SP	evstdwx	Vector Store Doubleword of Two Words Indexed
EVX	1000323			SP	evstdw	Vector Store Doubleword of Two Words
EVX	1000324			SP	evstdhx	Vector Store Doubleword of Four Halfwords Indexed
EVX	1000325			SP	evstdh	Vector Store Doubleword of Four Halfwords

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000330			SP	evstwhex	Vector Store Word of Two Halfwords from Even Indexed
EVX	1000331			SP	evstwhe	Vector Store Word of Two Halfwords from Even
EVX	1000334			SP	evstwhox	Vector Store Word of Two Halfwords from Odd Indexed
EVX	1000335			SP	evstwho	Vector Store Word of Two Halfwords from Odd
EVX	1000338			SP	evstwwex	Vector Store Word of Word from Even Indexed
EVX	1000339			SP	evstwwe	Vector Store Word of Word from Even
EVX	100033C			SP	evstwwox	Vector Store Word of Word from Odd Indexed
EVX	100033D			SP	evstwwo	Vector Store Word of Word from Odd
VX	1000340			VEC	vaddshs	Vector Add Signed Halfword Saturate
VX	1000342			VEC	vminsh	Vector Minimum Signed Halfword
VX	1000344			VEC	vsrah	Vector Shift Right Algebraic Word
VX	1000346			VEC	vcmpgtsh[.]	Vector Compare Greater Than Signed Halfword
VX	1000348			VEC	vmulesh	Vector Multiply Even Signed Halfword
VX	100034A			VEC	vcsxwfp	Vector Convert from Signed Fixed-Point Word to Single-Precision
VX	100034C			VEC	vspltish	Vector Splat Immediate Signed Halfword
VX	100034E			VEC	vupkhp	Vector Unpack High Pixel
X	1000358	SR		LIM	maclhw[o][.]	Multiply Accumulate Low Halfword to Word Modulo Signed
X	100035C	SR		LIM	nmaclhw[o][.]	Negative Multiply Accumulate Low Halfword to Word Modulo Signed
VX	1000380			VEC	vaddsws	Vector Add Signed Word Saturate
VX	1000382			VEC	vminsw	Vector Minimum Signed Word
VX	1000384			VEC	vsraw	Vector Shift Right Algebraic Word
VX	1000386			VEC	vcmpgtsw[.]	Vector Compare Greater Than Signed Word
VX	100038A			VEC	vcfpuxws	Vector Convert from Single-Precision to Unsigned Fixed-Point Word Saturate
VX	100038C			VEC	vspltisw	Vector Splat Immediate Signed Word
X	1000398	SR		LIM	maclhwsu[o][.]	Multiply Accumulate Low Halfword to Word Saturate Unsigned
VX	10003C6			VEC	vcmpbfp[.]	Vector Compare Bounds Single-Precision
VX	10003CA			VEC	vcfpsxws	Vector Convert from Single-Precision to Signed Fixed-Point Word Saturate
VX	10003CE			VEC	vupklp	Vector Unpack Low Pixel

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	10003D8	SR		LIM	maclhws[o][.]	Multiply Accumulate Low Halfword to Word Saturate Signed
X	10003DC	SR		LIM	nmaclhws[o][.]	Negative Multiply Accumulate Low Halfword to Word Saturate Signed
VX	1000400			VEC	vsububm	Vector Subtract Unsigned Byte Modulo
VX	1000402			VEC	vavgub	Vector Average Unsigned Byte
EVX	1000403			SP	evmhessf	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional
VX	1000404			VEC	vand	Vector AND
EVX	1000407			SP	evmhossf	Vector Multiply Halfwords, Odd, Signed, Fractional
EVX	1000408			SP	evmheumi	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer
EVX	1000409			SP	evmhesmi	Vector Multiply Halfwords, Even, Signed, Modulo, Integer
VX	100040A			VEC	vmaxfp	Vector Maximum Single-Precision
EVX	100040B			SP	evmhesmf	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional
EVX	100040C			SP	evmhoumi	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer
VX	100040C			VEC	vslo	Vector Shift Left by Octet
EVX	100040D			SP	evmosmi	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer
EVX	100040F			SP	evmosmf	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional
EVX	1000423			SP	evmhessfa	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional to Accumulator
EVX	1000427			SP	evmhossfa	Vector Multiply Halfwords, Odd, Signed, Fractional to Accumulator
EVX	1000428			SP	evmheumia	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer to Accumulator
EVX	1000429			SP	evmhesmia	Vector Multiply Halfwords, Even, Signed, Modulo, Integer to Accumulator
EVX	100042B			SP	evmhesmfa	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional to Accumulate
EVX	100042C			SP	evmhoumia	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer to Accumulator
EVX	100042D			SP	evmosmia	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer to Accumulator
EVX	100042F			SP	evmosmfa	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional to Accumulator

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	1000440			VEC	vsubuhm	Vector Subtract Unsigned Byte Modulo
VX	1000442			VEC	vavguh	Vector Average Unsigned Halfword
VX	1000444			VEC	vandc	Vector AND with Complement
EVX	1000447			SP	evmwhssf	Vector Multiply Word High Signed, Fractional
EVX	1000448			SP	evmwлумi	Vector Multiply Word Low Unsigned, Modulo, Integer
VX	100044A			VEC	vminfp	Vector Minimum Single-Precision
EVX	100044C			SP	evmwhumi	Vector Multiply Word High Unsigned, Modulo, Integer
VX	100044C			VEC	vsro	Vector Shift Right by Octet
EVX	100044D			SP	evmwhsmi	Vector Multiply Word High Signed, Modulo, Integer
EVX	100044F			SP	evmwhsmf	Vector Multiply Word High Signed, Modulo, Fractional
EVX	1000453			SP	evmwssf	Vector Multiply Word Signed, Saturate, Fractional
EVX	1000458			SP	evmwumi	Vector Multiply Word Unsigned, Modulo, Integer
EVX	1000459			SP	evmwsmi	Vector Multiply Word Signed, Modulo, Integer
EVX	100045B			SP	evmwsmf	Vector Multiply Word Signed, Modulo, Fractional
EVX	1000467			SP	evmwhssfa	Vector Multiply Word High Signed, Fractional to Accumulator
EVX	1000468			SP	evmwлумia	Vector Multiply Word Low Unsigned, Modulo, Integer to Accumulator
EVX	100046C			SP	evmwhumia	Vector Multiply Word High Unsigned, Modulo, Integer to Accumulator
EVX	100046D			SP	evmwhsmia	Vector Multiply Word High Signed, Modulo, Integer to Accumulator
EVX	100046F			SP	evmwhsmfa	Vector Multiply Word High Signed, Modulo, Fractional to Accumulator
EVX	1000473			SP	evmwssfа	Vector Multiply Word Signed, Saturate, Fractional to Accumulator
EVX	1000478			SP	evmwumia	Vector Multiply Word Unsigned, Modulo, Integer to Accumulator
EVX	1000479			SP	evmwsmia	Vector Multiply Word Signed, Modulo, Integer to Accumulator
EVX	100047B			SP	evmwsmfa	Vector Multiply Word Signed, Modulo, Fractional to Accumulator
VX	1000480			VEC	vsubuwm	Vector Subtract Unsigned Word Modulo
VX	1000482			VEC	vavguw	Vector Average Unsigned Word
VX	1000484			VEC	vor	Vector OR

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100004C0			SP	evaddusiaaw	Vector Add Unsigned, Saturate, Integer to Accumulator Word
EVX	100004C1			SP	evaddssiaaw	Vector Add Signed, Saturate, Integer to Accumulator Word
EVX	100004C2			SP	evsubfusiaaw	Vector Subtract Unsigned, Saturate, Integer to Accumulator Word
EVX	100004C3			SP	evsubfssiaaw	Vector Subtract Signed, Saturate, Integer to Accumulator Word
EVX	100004C4			SP	evmra	Initialize Accumulator
VX	100004C4			VEC	vxor	Vector XOR
EVX	100004C6			SP	evdivws	Vector Divide Word Signed
EVX	100004C7			SP	evdivwu	Vector Divide Word Unsigned
EVX	100004C8			SP	evaddumiaaw	Vector Add Unsigned, Modulo, Integer to Accumulator Word
EVX	100004C9			SP	evaddsmiaaw	Vector Add Signed, Modulo, Integer to Accumulator Word
EVX	100004CA			SP	evsubfumiaaw	Vector Subtract Unsigned, Modulo, Integer to Accumulator Word
EVX	100004CB			SP	evsubfsmiaaw	Vector Subtract Signed, Modulo, Integer to Accumulator Word
EVX	10000500			SP	evmheusiaaw	Vector Multiply Halfwords, Even, Unsigned, Saturate Integer and Accumulate into Words
EVX	10000501			SP	evmhessiaaw	Vector Multiply Halfwords, Even, Signed, Saturate, Integer and Accumulate into Words
VX	10000502			VEC	vavgsb	Vector Average Signed Byte
EVX	10000503			SP	evmhessfaaw	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional and Accumulate into Words
EVX	10000504			SP	evmhouisiaaw	Vector Multiply Halfwords, Odd, Unsigned, Saturate, Integer and Accumulate into Words
VX	10000504			VEC	vnor	Vector NOR
EVX	10000505			SP	evmhossiaaw	Vector Multiply Halfwords, Odd, Signed, Saturate, Integer and Accumulate into Words
EVX	10000507			SP	evmhossfaaw	Vector Multiply Halfwords, Odd, Signed, Saturate, Fractional and Accumulate into Words
EVX	10000508			SP	evmheumiaaw	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer and Accumulate into Words
EVX	10000509			SP	evmhesmiaaw	Vector Multiply Halfwords, Even, Signed, Modulo, Integer and Accumulate into Words

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000050B			SP	evmhesmfaaw	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional and Accumulate into Words
EVX	1000050C			SP	evmhoumiaaw	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer and Accumulate into Words
EVX	1000050D			SP	evmhosmiaaw	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer and Accumulate into Words
EVX	1000050F			SP	evmhosmfaaw	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional and Accumulate into Words
EVX	10000528			SP	evmhegumiaa	Vector Multiply Halfwords, Even, Guarded, Unsigned, Modulo, Integer and Accumulate
EVX	10000529			SP	evmhegsmiaa	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Integer and Accumulate
EVX	1000052B			SP	evmhegsmfaa	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Fractional and Accumulate
EVX	1000052C			SP	evmhogumiaa	Vector Multiply Halfwords, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate
EVX	1000052D			SP	evmhogsmiaa	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Integer and Accumulate
EVX	1000052F			SP	evmhogsmfaa	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Fractional and Accumulate
EVX	10000540			SP	evmwlusiaaw	Vector Multiply Word Low Unsigned Saturate, Integer and Accumulate into Words
EVX	10000541			SP	evmwlssiaaw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate into Words
VX	10000542			VEC	vavgsh	Vector Average Signed Halfword
EVX	10000544			SP	evmwhusiaaw	Vector Multiply Word High Unsigned, Integer and Accumulate into Words
EVX	10000547			SP	evmwhssfaaw	Vector Multiply Word High Signed, Fractional and Accumulate into Words
EVX	10000548			SP	evmwlumiaaw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate into Words
EVX	10000549			SP	evmwlsmiaaw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate into Words
EVX	1000054C			SP	evmwhumiaaw	Vector Multiply Word High Unsigned, Modulo, Integer and Accumulate into Words
EVX	1000054D			SP	evmwhsmiaaw	Vector Multiply Word High Signed, Modulo, Integer and Accumulate into Words
EVX	1000054F			SP	evmwhsmfaaw	Vector Multiply Word High Signed, Modulo, Fractional and Accumulate into Words

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	1000553			SP	evmwssfaa	Vector Multiply Word Signed, Saturate, Fractional and Accumulate
EVX	1000558			SP	evmwumiaa	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate
EVX	1000559			SP	evmwsmiaa	Vector Multiply Word Signed, Modulo, Integer and Accumulate
EVX	100055B			SP	evmwsmfaa	Vector Multiply Word Signed, Modulo, Fractional and Accumulate
EVX	1000580			SP	evmheusianw	Vector Multiply Halfwords, Even, Unsigned, Saturate Integer and Accumulate Negative into Words
VX	1000580			VEC	vsubcuw	Vector Subtract and Write Carry-Out Unsigned Word
EVX	1000581			SP	evmhessianw	Vector Multiply Halfwords, Even, Signed, Saturate, Integer and Accumulate Negative into Words
VX	1000582			VEC	vavgsw	Vector Average Signed Word
EVX	1000583			SP	evmhessfanw	Vector Multiply Halfwords, Even, Signed, Saturate, Fractional and Accumulate Negative into Words
EVX	1000584			SP	evmhousianw	Vector Multiply Halfwords, Odd, Unsigned, Saturate, Integer and Accumulate Negative into Words
EVX	1000585			SP	evmhossianw	Vector Multiply Halfwords, Odd, Signed, Saturate, Integer and Accumulate Negative into Words
EVX	1000587			SP	evmhossfanw	Vector Multiply Halfwords, Odd, Signed, Saturate, Fractional and Accumulate Negative into Words
EVX	1000588			SP	evmheumianw	Vector Multiply Halfwords, Even, Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	1000589			SP	evmhesmianw	Vector Multiply Halfwords, Even, Signed, Modulo, Integer and Accumulate Negative into Words
EVX	100058B			SP	evmhesmfanw	Vector Multiply Halfwords, Even, Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	100058C			SP	evmhoumianw	Vector Multiply Halfwords, Odd, Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	100058D			SP	evmosmianw	Vector Multiply Halfwords, Odd, Signed, Modulo, Integer and Accumulate Negative into Words
EVX	100058F			SP	evmosmfanw	Vector Multiply Halfwords, Odd, Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	10005A8			SP	evmhegumian	Vector Multiply Halfwords, Even, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
EVX	10005A9			SP	evmhegsmian	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Integer and Accumulate Negative

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
EVX	100005AB			SP	evmhgsmfan	Vector Multiply Halfwords, Even, Guarded, Signed, Modulo, Fractional and Accumulate Negative
EVX	100005AC			SP	evmhogumian	Vector Multiply Halfwords, Odd, Guarded, Unsigned, Modulo, Integer and Accumulate Negative
EVX	100005AD			SP	evmhogsmian	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Integer and Accumulate Negative
EVX	100005AF			SP	evmhogsmfan	Vector Multiply Halfwords, Odd, Guarded, Signed, Modulo, Fractional and Accumulate Negative
EVX	100005C0			SP	evmwlusianw	Vector Multiply Word Low Unsigned Saturate, Integer and Accumulate Negative into Words
EVX	100005C1			SP	evmwlsianw	Vector Multiply Word Low Signed, Saturate, Integer and Accumulate Negative into Words
EVX	100005C4			SP	evmwhusianw	Vector Multiply Word High Unsigned, Integer and Accumulate Negative into Words
EVX	100005C5			SP	evmwhssianw	Vector Multiply Word High Signed, Integer and Accumulate Negative into Words
EVX	100005C7			SP	evmwhssfanw	Vector Multiply Word High Signed, Fractional and Accumulate Negative into Words
EVX	100005C8			SP	evmwlumianw	Vector Multiply Word Low Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	100005C9			SP	evmwlsnianw	Vector Multiply Word Low Signed, Modulo, Integer and Accumulate Negative into Words
EVX	100005CC			SP	evmwhumianw	Vector Multiply Word High Unsigned, Modulo, Integer and Accumulate Negative into Words
EVX	100005CD			SP	evmwhsmianw	Vector Multiply Word High Signed, Modulo, Integer and Accumulate Negative into Words
EVX	100005CF			SP	evmwhsmfanw	Vector Multiply Word High Signed, Modulo, Fractional and Accumulate Negative into Words
EVX	100005D3			SP	evmwssfan	Vector Multiply Word Signed, Saturate, Fractional and Accumulate Negative
EVX	100005D8			SP	evmwumian	Vector Multiply Word Unsigned, Modulo, Integer and Accumulate Negative
EVX	100005D9			SP	evmwsmian	Vector Multiply Word Signed, Modulo, Integer and Accumulate Negative
EVX	100005DB			SP	evmwsmfan	Vector Multiply Word Signed, Modulo, Fractional and Accumulate Negative
VX	10000600			VEC	vsububs	Vector Subtract Unsigned Byte Saturate
VX	10000604			VEC	mfvscr	Move from Vector Status and Control Register
VX	10000608			VEC	vsum4ubs	Vector Sum across Quarter Unsigned Byte Saturate

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
VX	1000640			VEC	vsubuhs	Vector Subtract Unsigned Halfword Saturate
VX	1000644			VEC	mtvscr	Move to Vector Status and Control Register
VX	1000648			VEC	vsum4shs	Vector Sum across Quarter Signed Halfword Saturate
VX	1000680			VEC	vsubuws	Vector Subtract Unsigned Word Saturate
VX	1000688			VEC	vsum2sws	Vector Sum across Half Signed Word Saturate
VX	1000700			VEC	vsubsbs	Vector Subtract Signed Byte Saturate
VX	1000708			VEC	vsum4sbs	Vector Sum across Quarter Signed Byte Saturate
VX	1000740			VEC	vsubshs	Vector Subtract Signed Halfword Saturate
VX	1000780			VEC	vsubsws	Vector Subtract Signed Word Saturate
VX	1000788			VEC	vsumsws	Vector Sum across Signed Word Saturate
D8	18000000			VLE	e_lbzu	Load Byte and Zero with Update
D8	18000100			VLE	e_lhzu	Load Halfword and Zero with Update
D8	18000200			VLE	e_lwzu	Load Word and Zero with Update
D8	18000300			VLE	e_lhau	Load Halfword Algebraic with Update
D8	18000400			VLE	e_stbu	Store Byte with Update
D8	18000500			VLE	e_sthu	Store Halfword with Update
D8	18000600			VLE	e_stwu	Store word with Update
D8	18000800			VLE	e_lmw	Load Multiple Word
D8	18000900			VLE	e_stmw	Store Multiple Word
SC18	18008000	SR		VLE	e_addi[.]	Add Scaled Immediate
SC18	18009000	SR		VLE	e_addic[.]	Add Scaled Immediate Carrying
SC18	1800A000			VLE	e_mulli	Multiply Low Scaled Immediate
SC18	1800A800			VLE	e_cmpi	Compare Scaled Immediate Word
SC18	1800B000	SR		VLE	e_subfic[.]	Subtract From Scaled Immediate Carrying
SC18	1800C000	SR		VLE	e_andi[.]	AND Scaled Immediate
SC18	1800D000	SR		VLE	e_ori[.]	OR Scaled Immediate
SC18	1800E000	SR		VLE	e_xori[.]	XOR Scaled Immediate
SC18	1880A800			VLE	e_cmpli	Compare Logical Scaled Immediate Word
D	1C000000			VLE	e_add16i	Add Immediate
OIM5	2000----			VLE	se_addi	Add Immediate Short Form
OIM5	2200----			VLE	se_cmpli	Compare Logical Immediate Word
OIM5	2400----	SR		VLE	se_subi[.]	Subtract Immediate

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
IM5	2A00----			VLE	se_cmpi	Compare Immediate Word Short Form
IM5	2C00----			VLE	se_bmaski	Bit Mask Generate Immediate
IM5	2E00----			VLE	se_andi	AND Immediate Short Form
D	30000000			VLE	e_lbz	Load Byte and Zero
D	34000000			VLE	e_stb	Store Byte
D	38000000			VLE	e_lha	Load Halfword Algebraic
RR	4000----			VLE	se_srw	Shift Right Word
RR	4100----	SR		VLE	se_sraw	Shift Right Algebraic Word
RR	4200----			VLE	se_slw	Shift Left Word
RR	4400----			VLE	se_or	OR Short Form
RR	4500----			VLE	se_andc	AND with Complement Short Form
RR	4600----	SR		VLE	se_and[.]	AND Short Form
IM7	4800----			VLE	se_li	Load Immediate Short Form
D	50000000			VLE	e_lwz	Load Word and Zero
D	54000000			VLE	e_stw	Store Word
D	58000000			VLE	e_lhz	Load Halfword and Zero
D	5C000000			VLE	e_sth	Store Halfword
IM5	6000----			VLE	se_bclri	Bit Clear Immediate
IM5	6200----			VLE	se_bgeni	Bit Generate Immediate
IM5	6400----			VLE	se_bseti	Bit Set Immediate
IM5	6600----			VLE	se_btsti	Bit Test Immediate
IM5	6800----			VLE	se_srwi	Shift Right Word Immediate Short Form
IM5	6A00----	SR		VLE	se_srawi	Shift Right Algebraic Immediate
IM5	6C00----			VLE	se_slwi	Shift Left Word Immediate Short Form
LI20	70000000			VLE	e_li	Load Immediate
I16A	70008800	SR		VLE	e_add2i.	Add (2 operand) Immediate and Record
I16A	70009000			VLE	e_add2is	Add (2 operand) Immediate Shifted
IA16	70009800			VLE	e_cmp16i	Compare Immediate Word
I16A	7000A000			VLE	e_mull2i	Multiply (2 operand) Low Immediate
I16A	7000A800			VLE	e_cmpl16i	Compare Logical Immediate Word
IA16	7000B000			VLE	e_cmph16i	Compare Halfword Immediate
IA16	7000B800			VLE	e_cmphl16i	Compare Halfword Logical Immediate

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
I16L	7000C000			VLE	e_or2i	OR (2operand) Immediate
I16L	7000C800	SR		VLE	e_and2i.	AND (2 operand) Immediate
I16L	7000D000			VLE	e_or2is	OR (2 operand) Immediate Shifted
I16L	7000E000			VLE	e_lis	Load Immediate Shifted
I16L	7000E800	SR		VLE	e_and2is.	AND (2 operand) Immediate Shifted
M	74000000			VLE	e_rlwimi	Rotate Left Word Immediate then Mask Insert
M	74000001			VLE	e_rlwinm	Rotate Left Word Immediate then AND with Mask
BD24	78000000			VLE	e_b[l]	Branch [and Link]
BD15	7A000000	CT		VLE	e_bc[l]	Branch Conditional [and Link]
X	7C000000			B	cmp	Compare
X	7C000008			B	tw	Trap Word
X	7C00000C			VEC	lvsl	Load Vector for Shift Left Indexed
X	7C00000E			VEC	lvebx	Load Vector Element Byte Indexed
XO	7C000010	SR		B	subfc[o][.]	Subtract From Carrying
XO	7C000012	SR		64	mulhdu[.]	Multiply High Doubleword Unsigned
XO	7C000014			B	addc[o][.]	Add Carrying
XO	7C000016	SR		B	mulhwu[.]	Multiply High Word Unsigned
X	7C00001C			VLE	e_cmph	Compare Halfword
A	7C00001E			B	isel	Integer Select
XL	7C000020			VLE	e_mcrf	Move CR Field
AFX	7C000026			B	mfcrr	Move From Condition Register
X	7C000028			B	lwarx	Load Word and Reserve Indexed
X	7C00002A			64	ldx	Load Doubleword Indexed
X	7C00002C			E	icbt	Instruction Cache Block Touch
X	7C00002E			B	lwzx	Load Word and Zero Indexed
X	7C000030	SR		B	slw[.]	Shift Left Word
X	7C000034	SR		B	cntlzw[.]	Count Leading Zeros Word
X	7C000036	SR		64	sld[.]	Shift Left Doubleword
X	7C000038	SR		B	and[.]	AND
X	7C00003A		P	E.PD	ldepix	Load Doubleword by External Process ID Indexed
X	7C00003E		P	E.PD	lwepix	Load Word by External Process ID Indexed
X	7C000040			B	cmpl	Compare Logical

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
XL	7C000042			VLE	e_crnor	Condition Register NOR
X	7C00004C			VEC	lvsr	Load Vector for Shift Right Indexed
X	7C00004E			VEC	lvehx	Load Vector Element Halfword Indexed
XO	7C000050	SR		B	subf[o][.]	Subtract From
X	7C00005C			VLE	e_cmphi	Compare Halfword Logical
X	7C00006A			64	ldux	Load Doubleword with Update Indexed
X	7C00006C			B	dcbst	Data Cache Block Store
X	7C00006E			B	lwzux	Load Word and Zero with Update Indexed
X	7C000070	SR		VLE	e_slwi[.]	Shift Left Word Immediate
X	7C000074	SR		64	cntlzd[.]	Count Leading Zeros Doubleword
X	7C000078	SR		B	andc[.]	AND with Complement
X	7C00007C			WT	wait	Wait
X	7C000088			64	td	Trap Doubleword
X	7C00008E			VEC	lvewx	Load Vector Element Word Indexed
XO	7C000092	SR		64	mulhd[.]	Multiply High Doubleword
XO	7C000096	SR		B	mulhw[.]	Multiply High Word
X	7C0000A6		P	B	mfmsr	Move From Machine State Register
X	7C0000A8			64	ldarx	Load Doubleword and Reserve Indexed
X	7C0000AC			B	dcbf	Data Cache Block Flush
X	7C0000AE			B	lbzx	Load Byte and Zero Indexed
X	7C0000BE		P	E.PD	lbepx	Load Byte by External Process ID Indexed
X	7C0000CE			VEC	lvx[l]	Load Vector Indexed [Last]
X	7C0000D0	SR		B	neg[o][.]	Negate
X	7C0000EE			B	lbzux	Load Byte and Zero with Update Indexed
X	7C0000F4			B	popcntb	Population Count Bytes
X	7C0000F8	SR		B	nor[.]	NOR
X	7C0000FE		P	E.PD	dcbfep	Data Cache Block Flush by External Process ID
XL	7C000102			VLE	e_crandc	Condition Register AND with Complement
X	7C000106		P	E	wrtee	Write MSR External Enable
X	7C00010C		M	E.CL	dcbtstls	Data Cache Block Touch for Store and Lock Set
VX	7C00010E			VEC	stvebx	Store Vector Element Byte Indexed
XO	7C000110	SR		B	subfe[o][.]	Subtract From Extended

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
XO	7C000114	SR		B	adde[o][.]	Add Extended
EVX	7C00011D		P	E.PD	evlddepX	Vector Load Doubleword into Doubleword by External Process ID Indexed
XFX	7C000120			B	mtrf	Move to Condition Register Fields
X	7C000124		P	E	mtmsr	Move To Machine State Register
X	7C00012A			64	stdX	Store Doubleword Indexed
X	7C00012D			B	stwcX.	Store Word Conditional Indexed
X	7C00012E			B	stwX	Store Word Indexed
X	7C00013A		P	E.PD	stdepX	Store Doubleword by External Process ID Indexed
X	7C00013E		P	E.PD	stwepX	Store Word by External Process ID Indexed
X	7C000146		P	E	wrteei	Write MSR External Enable Immediate
X	7C00014C		M	E.CL	dcbtls	Data Cache Block Touch and Lock Set
VX	7C00014E			VEC	stvehX	Store Vector Element Halfword Indexed
X	7C00016A			64	stdux	Store Doubleword with Update Indexed
X	7C00016E			B	stwux	Store Word with Update Indexed
XL	7C000182			VLE	e_crxor	Condition Register XOR
VX	7C00018E			VEC	stvewX	Store Vector Element Word Indexed
XO	7C000190	SR		B	subfze[o][.]	Subtract From Zero Extended
XO	7C000194	SR		B	addze[o][.]	Add to Zero Extended
X	7C00019C		P	E.PC	msgsnd	Message Send
EVX	7C00019D		P	E.PD	evstddepX	Vector Store Doubleword into Doubleword by External Process ID Indexed
X	7C0001AD			64	stdcX.	Store Doubleword Conditional Indexed
X	7C0001AE			B	stbX	Store Byte Indexed
X	7C0001BE		P	E.PD	stbepX	Store Byte by External Process ID Indexed
XL	7C0001C2			VLE	e_crnand	Condition Register NAND
X	7C0001CC		M	E.CL	icblc	Instruction Cache Block Lock Clear
VX	7C0001CE			VEC	stvx[l]	Store Vector Indexed [Last]
XO	7C0001D0	SR		B	subfme[o][.]	Subtract From Minus One Extended
XO	7C0001D2	SR		64	mulld[o][.]	Multiply Low Doubleword
XO	7C0001D4	SR		B	addme[o][.]	Add to Minus One Extended
XO	7C0001D6	SR		B	mulw[o][.]	Multiply Low Word
X	7C0001DC		P	E.PC	msgclr	Message Clear

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C0001EC			B	dcbtst	Data Cache Block Touch for Store
X	7C0001EE			B	stbux	Store Byte with Update Indexed
X	7C0001FE		P	E.PD	dcbtstep	Data Cache Block Touch for Store by External Process ID
XL	7C000202			VLE	e_crand	Condition Register AND
AFX	7C000206		P	E	mfdcrx	Move From Device Control Register Indexed
X	7C00020E		P	E.PD	lvepxl	Load Vector by External Process ID Indexed LRU
XO	7C000214			B	add[o][.]	Add
X	7C00022C			B	dcbt	Data Cache Block Touch
X	7C00022E			B	lhzx	Load Halfword and Zero Indexed
X	7C000230	SR		VLE	e_rlw[.]	Rotate Left Word
X	7C000238	SR		B	eqv[.]	Equivalent
X	7C00023E		P	E.PD	lhexp	Load Halfword by External Process ID Indexed
XL	7C000242			VLE	e_creqv	Condition Register Equivalent
AFX	7C000246		P	E	mfdcrux	Move From Device Control Register User Indexed
X	7C00024E		P	E.PD	lvepx	Load Vector by External Process ID Indexed
X	7C00026E			B	lhzux	Load Halfword and Zero with Update Indexed
X	7C000270	SR		VLE	e_rlwi[.]	Rotate Left Word Immediate
D	7C000278	SR		B	xor[.]	XOR
X	7C00027E		P	E.PD	dcbtstep	Data Cache Block Touch by External Process ID
AFX	7C000286		P	E	mfdcr	Move From Device Control Register
X	7C00028C		P	E.CD	dcread	Data Cache Read
AFX	7C00029C		O	E.PM	mfpmr	Move From Performance Monitor Register
AFX	7C0002A6		O	B	mfspr	Move From Special Purpose Register
X	7C0002AA			64	lwax	Load Word Algebraic Indexed
X	7C0002AE			B	lhax	Load Halfword Algebraic Indexed
X	7C0002EA			64	lwaux	Load Word Algebraic with Update Indexed
X	7C0002EE			B	lhaux	Load Halfword Algebraic with Update Indexed
X	7C000306		P	E	mtdcrx	Move To Device Control Register Indexed
X	7C00030C		M	E.CL	dcblc	Data Cache Block Lock Clear
X	7C00032E			B	sthx	Store Halfword Indexed
X	7C000338	SR		B	orc[.]	OR with Complement
X	7C00033E		P	E.PD	sthepx	Store Halfword by External Process ID Indexed

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
XL	7C000342			VLE	e_crorc	Condition Register OR with Complement
X	7C000346			E	mtdcruz	Move To Device Control Register User Indexed
X	7C00036E			B	sthux	Store Halfword with Update Indexed
X	7C000378	SR		B	or[.]	OR
XL	7C000382			VLE	e_cror	Condition Register OR
XFX	7C000386		P	E	mtdcr	Move To Device Control Register
X	7C00038C		P	E.CI	dci	Data Cache Invalidate
XO	7C000392	SR		64	divdu[o][.]	Divide Doubleword Unsigned
XO	7C000396	SR		B	divwu[o][.]	Divide Word Unsigned
XFX	7C00039C		O	E.PM	mtpmr	Move To Performance Monitor Register
XFX	7C0003A6		O	B	mtspr	Move To Special Purpose Register
X	7C0003AC		P	E	dcbi	Data Cache Block Invalidate
X	7C0003B8	SR		B	nand[.]	NAND
X	7C0003CC		M	E.CL	icbtls	Instruction Cache Block Touch and Lock Set
X	7C0003CC		P	E.CD	dcread	Data Cache Read
XO	7C0003D2	SR		64	divd[o][.]	Divide Doubleword
XO	7C0003D6	SR		B	divw[o][.]	Divide Word
X	7C000400			B	mcrxr	Move To Condition Register From XER
X	7C00042A			MA	lswx	Load String Word Indexed
X	7C00042C			B	lwbrx	Load Word Byte-Reversed Indexed
X	7C000430	SR		B	srw[.]	Shift Right Word
X	7C000436	SR		64	srd[.]	Shift Right Doubleword
X	7C00046C		P	E	tlbsync	TLB Synchronize
X	7C000470	SR		VLE	e_srwi[.]	Shift Right Word Immediate
X	7C0004AA			MA	lswi	Load String Word Immediate
X	7C0004AC			B	sync	Synchronize
X	7C0004BE		P	E.PD	lfdep	Load Floating-Point Double by External Process ID Indexed
X	7C00052A			MA	stswx	Store String Word Indexed
X	7C00052C			B	stwbrx	Store Word Byte-Reversed Indexed
X	7C0005AA			MA	stswi	Store String Word Immediate
X	7C0005BE		P	E.PD	stfdep	Store Floating-Point Double by External Process ID Indexed

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
X	7C0005EC			E	dcba	Data Cache Block Allocate
X	7C00060E		P	E.PD	stvepxl	Store Vector by External Process ID Indexed LRU
X	7C000624		P	E	tlbivax	TLB Invalidate Virtual Address Indexed
X	7C00062C			B	lhbrx	Load Halfword Byte-Reversed Indexed
X	7C000630	SR		B	sraw[.]	Shift Right Algebraic Word
X	7C000634	SR		64	srad[.]	Shift Right Algebraic Doubleword
X	7C00064E		P	E.PD	stvepx	Store Vector by External Process ID Indexed
X	7C000670	SR		B	srawi[.]	Shift Right Algebraic Word Immediate
X	7C000674	SR		64	sradi[.]	Shift Right Algebraic Doubleword Immediate
AFX	7C0006AC			E	mbar	Memory Barrier
X	7C000724		P	E	tlbsx	TLB Search Indexed
X	7C00072C			B	sthbrx	Store Halfword Byte-Reversed Indexed
X	7C000734	SR		B	extsh[.]	Extend Sign Halfword
X	7C000764		P	E	tlbre	TLB Read Entry
X	7C000774	SR		B	extsb[.]	Extend Sign Byte
X	7C00078C		P	E.CI	ici	Instruction Cache Invalidate
X	7C0007A4		P	E	tlbwe	TLB Write Entry
X	7C0007AC			B	icbi	Instruction Cache Block Invalidate
X	7C0007B4	SR		64	extsw[.]	Extend Sign Word
X	7C0007BE		P	E.PD	icbiep	Instruction Cache Block Invalidate by External Process ID
X	7C0007CC		P	E.CD	icread	Instruction Cache Read
X	7C0007EC			B	dcbz	Data Cache Block set to Zero
X	7C0007FE		P	E.PD	dcbzep	Data Cache Block set to Zero by External Process ID
AFX	7C100026			B	mfocrf	Move From One Condition Register Field
AFX	7C100120			B	mtocrf	Move To One Condition Register Field
SD4	8000----			VLE	se_lbz	Load Byte and Zero Short Form
SD4	9000----			VLE	se_stb	Store Byte Short Form
SD4	A000----			VLE	se_lhz	Load Halfword and Zero Short Form
SD4	B000----			VLE	se_sth	Store Halfword Short Form
SD4	C000----			VLE	se_lwz	Load Word and Zero Short Form
SD4	D000----			VLE	se_stw	Store Word Short Form

Table B-3. VLE Instruction Set Sorted by Opcode

Form	Opcode (hexadecimal) ¹	Mode Dep. ²	Priv ²	Cat ²	Mnemonic	Instruction
BD8	E000----			VLE	se_bc	Branch Conditional Short Form
BD8	E800----			VLE	se_b[l]	Branch [and Link]

¹ For 16-bit instructions, this column represents the 16-bit hexadecimal instruction encoding with the opcode and extended opcode in the corresponding fields in the instruction, and with 0s in bit positions that are not opcode bits; dashes are used following the opcode to indicate the form is a 16-bit instruction. For 32-bit instructions, this column represents the 32-bit hexadecimal instruction encoding with the opcode and extended opcode in the corresponding fields in the instruction, and with 0s in bit positions that are not opcode bits.

² ¹See the key to the mode dependency and privilege column in [Table B-1](#).

