Freescale Semiconductor
Engineering Bulletin

Document Number: EB696
Rev. 0, 07/2008

New VLE Instructions for Improving Interrupt Handler Efficiency

by: Robert Moran
Applications Engineer
Microcontroller Solutions Group

1 Introduction

The VLE instruction set provides a mixture of 16 and 32-bit instructions that are used in the Qorivva MPC55xx, MPC551x and MPC56xx, product families. This engineering bulletin details new VLE instructions that have been implemented to improve the efficiency of handling interrupts.

These new instructions allow groups of volatile registers to be saved to or restored from the stack using a single instruction. These instructions allow for simpler coding within the interrupt handler, in some cases, quicker context save, and restore times.

Contents

2 Devices Affected by New Instructions

All devices that implement the e200z1 and e200z0 cores support these new instructions in the VLE instruction set.

These instructions are currently implemented on the following devices. See Table 1.

Table 1. Devices Supporting New Instructions

Device	Core	New VLE Instruction Support
MPC51xx	e200z1 and e200z0	Yes
MPC563xM	e200z335	Yes
MPC560xB	e200z0	Yes
MPC560xP	e200z0	Yes
MPC560xS	e200z0	Yes

3 Instruction Operation Codes

The operation codes of the additional instructions are listed in Table 2.

Table 2. Operation Codes of New Instructions

Instruction Bit Coding	0…..5	6…..10	11…..15	16…..23	24…..31
e_ldmvgprw	6 (0b0001_10)	0b00000	RA	0b0001_0000	D8
e_stmvgprw	6 (0b0001_10)	0b00000	RA	0b0001_0001	D8
e_ldmvsprw	6 (0b0001_10)	0b00001	RA	0b0001_0000	D8
e_stmvsprw	6 (0b0001_10)	0b00001	RA	0b0001_0001	D8
e_ldmvsrrw	6 (0b0001_10)	0b00100	RA	0b0001_0000	D8
e_stmvsrrw	6 (0b0001_10)	0b00100	RA	0b0001_0001	D8
e_ldmvcsrrw	6 (0b0001_10)	0b00101	RA	0b0001_0000	D8
e_stmvcsrrw	6 (0b0001_10)	0b00101	RA	0b0001_0001	D8
e_ldmvdsrrw	6 (0b0001_10)	0b00110	RA	0b0001_0000	D8
e_stmvdsrrw	6 (0b0001_10)	0b00110	RA	0b0001_0001	D8

NOTE

The 8-bit D8 field is sign-extended and added to the contents of the GPR designated by RA to produce the effective load or store address.

4 Instruction Descriptions

The embedded application binary interface (EABI) defines specific registers to be classified as volatile and non-volatile registers. These registers are shown in Table 3.

Table 3. EABI Register Definitions

r0	function linkage	volatile	r31	local variables / env pointer	non-volatile
r1	stack frame pointer	dedicated	cr0-cr1	condition reg. fields	volatile
r2	small data area 2 pointer	dedicated	cr2-cr4	condition reg. fields	non-volatile
r3-r4	parameters/return values	volatile	cr5-cr7	condition reg. fields	volatile
r5-r10	parameters	volatile	lr	link register	volatile
r11-r12	function linkage	volatile	ctr	count register	volatile
r13	small data area pointer	dedicated	xer	integer exception reg.	volatile
r14-r30	local variables	non-volatile			

Using a typical VLE interrupt handler, the volatile registers are saved and restored from the stack using store and load instructions for each individual register. This is shown in Figure 1.

```
mfSRR1 r0                       # Store SRR1 (must be done before enabling EE)
e_stw    r0, 0x10 (r1)
mfSRR0 r0                       # Store SRR0 (must be done before enabling EE)
e_stw    r0, 0x0C (r1)
mfLR   r0                       # Store LR (Store now since LR will be
                                # used for ISR Vector)
wrteei 1                        # Set MSR[EE]=1

e_stw    r12, 0x4C (r1)         # Store GPRs
e_stw    r11, 0x48 (r1)
e_stw    r10, 0x44 (r1)
e_stw    r9,  0x40 (r1)
e_stw    r8,  0x3C (r1)
e_stw    r7,  0x38 (r1)
e_stw    r6,  0x34 (r1)
e_stw    r5,  0x30 (r1)
e_stw    r4,  0x2c (r1)
e_stw    r3,  0x28 (r1)
e_stw    r0,  0x24 (r1)

mfCR   r0                        # Store CR
e_stw    r0,  0x20 (r1)
mfXER  r0                        # Store XER
e_stw    r0,  0x1C (r1)
mfCTR  r0                        # Store CTR
e_stw    r0,  0x18 (r1)
```

Figure 1. Interrupt Handler Prolog

The new VLE instructions allow groups of volatile registers to be saved and restored to the stack using a single instruction. This reduces the number of instructions required and the time taken to save or restore all the volatile registers. Table 4 describes how the new VLE instructions relate to the groups of volatile registers.

Table 4. Instruction Overview

Instruction	Description	Registers Written / Read
e_ldmvgprw	Load multiple volatile general purpose registers (GPR)	r0, r3:r12
e_stmvgprw	Store multiple volatile general purpose registers (GPR)	r0, r3:r12
e_ldmvsprw	Load multiple volatile special purpose registers (SPR)	CR, LR, CTR, and XER
e_stmvsprw	Store multiple volatile special purpose registers (SPR)	CR, LR, CTR, and XER
e_ldmvsrrw	Load multiple volatile save and restore registers (SSR)	SRR0, SRR1
e_stmvsrrw	Store multiple volatile save and restore registers (SSR)	SRR0, SRR1
e_ldmvcsrrw	Load multiple volatile critical save and restore registers (CSSR)	CSRR0, CSRR1
e_stmvcsrrw	Store multiple volatile critical save and restore registers (CSSR)	CSRR0, CSRR1
e_ldmvdsrrw	Load multiple volatile debug save and restore registers (DRRR)	DSRR0, DSRR1
e_stmvdsrrw	Store multiple volatile debug save and restore registers (DSSR)	DSRR0, DSRR1

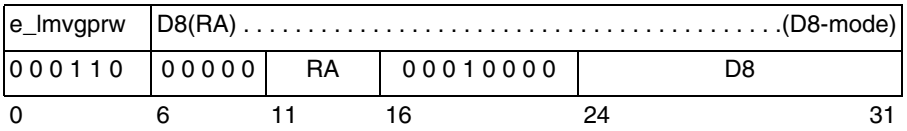
Figure 2 illustrates how these new instructions are implemented in an interrupt handler prolog.

```
e_stmvsrrw 0x04 (r1)              # Store SRRs (SRR0, SRR1) (must be done before
                                  # enabling EE)
e_stmvsprw  0x38 (r1)             # Store SRRs (CR,LR,CTR,XER)

wrteei                            # Set MSR[EE]=1

e_stmvgprw  0x0C (r1)             # Store volatile GPRs (R0, R3-R12)
```

Figure 2. Interrupt Handler Prolog Using New Instructions

5 Instruction Listings

5.1 Load Multiple Volatile GPR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

GPR(r0)32:63  ←  MEM(EA,4)
EA ← (EA+4)

r ← 3
do while r ≤ 12
 GPR(r)32:63 ← MEM(EA,4)

EA ← (EA,4)
r ← r + 1
```

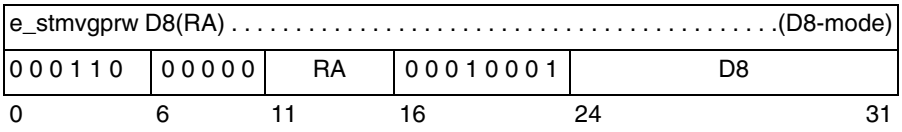
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers GPR(R0), and GPR(R3) through GPR(12) are loaded from n consecutive words in storage 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

5.2 Store Multiple Volatile GPR Word




```
if RA=0 then EA ←  EXTS(D8)
else         EA ←(GPR(RA)+EXTS(D8))

MEM(EA,4) ←  GPR(r0)32:63
EA ←(EA+4)

R ←  3
do while r ≤ 12
MEM(EA,4) ←  GPR(r)32:63
r ← r+1
EA ←(EA+4)
```

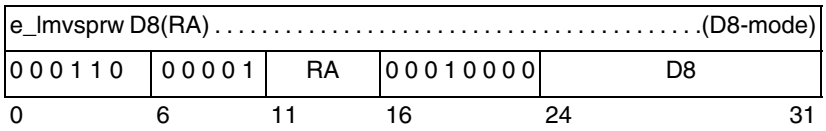
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers GPR(R0), and GPR(R3) through GPR(12) are stored in n consecutive words in storage 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

5.3 Load Multiple Volatile SPR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

CR32:63 ← MEM(EA,4)
EA ← (EA+4)

LR32:63 ← MEM(EA,4)
EA ← (EA+4)

CTR32:63 ← MEM(EA,4)
EA ← (EA+4)

XER32:63 ← MEM(EA,4)
```

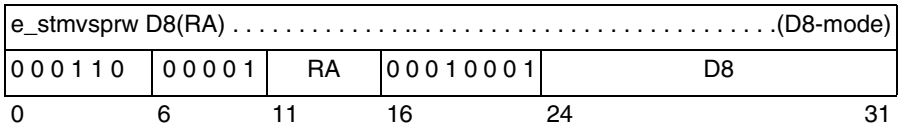
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers CR, LR, CTR, and XER are loaded from n consecutive words in storage 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: CR, LR, CTR, XER

5.4 Store Multiple Volatile SPR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

MEM(EA,4) ← CR32:63
EA ← (EA+4)

MEM(EA,4) ← LR32:63
EA ← (EA+4)

MEM(EA,4) ← CTR32:63
EA ← (EA+4)

MEM(EA,4) ← XER32:63
```

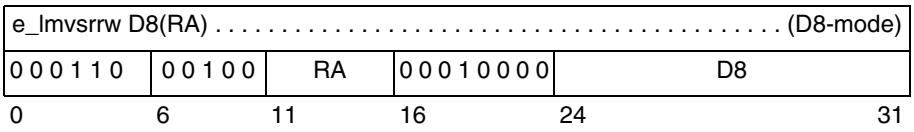
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers CR, LR, CTR, and XER are stored in n consecutive words in storage 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

5.5 Load Multiple Volatile SRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

SRR032:63 ← MEM(EA,4)
EA ← (EA+4)
SRR132:63 ← MEM(EA,4)
```

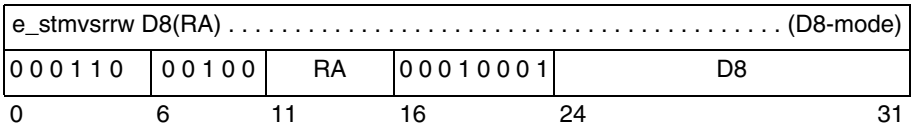
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers SRR0 and SRR1 are loaded from consecutive words in storage 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: SRR0, SRR1

5.6 Store Multiple Volatile SRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

MEM(EA,4) ← SRR0[32:63]
EA ← (EA+4)
MEM(EA,4) ← SRR1[32:63]
```

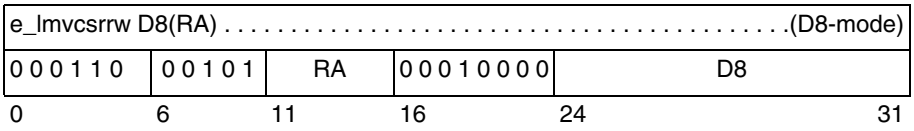
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers SRR0 and SRR1 are stored in consecutive words in storage 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

5.7 Load Multiple Volatile CSRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

CSRR0[32:63] ← MEM(EA,4)
EA ← (EA+4)
CSRR1[32:6]3 ← MEM(EA,4)
```

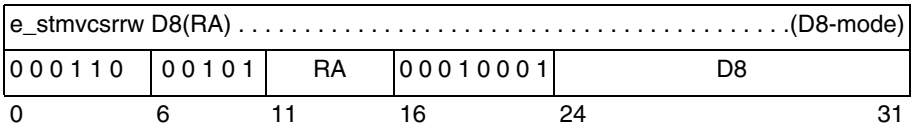
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers CSRR0 and CSRR1 are loaded from consecutive words in storage 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: CSRR0, CSRR

5.8 Store Multiple Volatile CSRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

MEM(EA,4) ← CSRR032:63
EA ← (EA+4)
MEM(EA,4) ← CSRR132:63
```

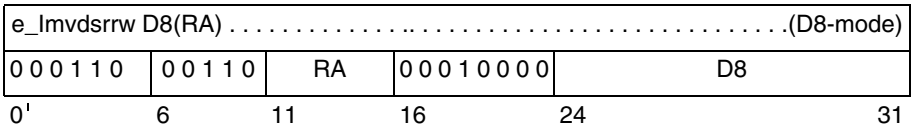
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers CSRR0 and CSRR1 are stored in consecutive words in storage 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

5.9 Load Multiple Volatile DSRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

DSRR0 32:63 ← MEM(EA,4)
EA ← (EA+4)
DSRR1 32:63 ← MEM(EA,4)
```

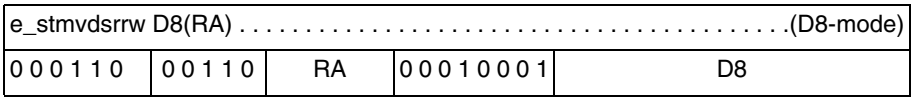
Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers DSRR0 and DSRR1 are loaded from consecutive words in storage 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: DSRR0, DSRR1

5.10 Store Multiple Volatile DSRR Word




```
if RA=0 then EA ← EXTS(D8)
else         EA ← (GPR(RA)+EXTS(D8))

MEM(EA,4) ← DSRR0[32:63]
EA ← (EA+4)
MEM(EA,4) ← DSRR1[32:63]
```

Let the effective address (EA) be the sum of the contents of GPR(RA), or 0 if RA=0, and the sign-extended value of the D8 instruction field.

Bits 32:63 of registers DSRR0 and DSRR1 are stored in consecutive words in storage 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

6 Revision history

Table 5. Changes made April 2012[1]

Section	Description
Front page	Add SafeAssure branding.
1	Add Qorivva branding.
Back page	Apply new back page format.

[1] No substantive changes were made to the content of this document; therefore the revision number was not incremented.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: EB696
Rev. 0
07/2008