Technical note

SPC58x IVORs – List of exceptions and triggers

Introduction

Power PC architecture implements various kinds of exception, which allow to save the core state, react to asynchronous event or failure condition and then restore and continue the previous context.

The exception mechanism also allows the processor to escalate to supervisor state.

The aim of this technical note is to describe the type of IVOR exceptions, their sources and how to trigger them.

1 General introduction

Power PC cores support the extended exception-handling model with the capability of nesting and vector programmability. External interrupts1 and error conditions are mapped to interrupt vectors (IVORs) based on their character. Some IVORs combine multiple sources and the handler has to check for the exact failure condition by reading flags in the relevant registers.

1.1 IVORs list

The following sections give the z4 type of core, which is used across the SPC58 family.

1.1.1 IVOR0: critical input

This exception is raised if the signal *p_critint_b* is asserted (level sensitive).

Connection of this signal is device specific, in our case it is one of some possible targets from NMI inputs. Critical input is maskable in the MSR register.

1.1.2 IVOR1: machine check

IVOR1 exception combines multiple sources. This exception is raised if:

- p_mcp_b signal is asserted (edge sensitive) (“Machine check request”–input)
- p_nmi_b signal is asserted (edge sensitive) (“NMI exception request”–input)

The connection of these inputs is device specific, in our case it is one of some possible targets from NMI inputs. NMI is a nonmaskable input. MCP is maskable in the *HID0* register.

- p_mcp_out signal is asserted (error condition occurred–output)

Error output signal is asserted in cases where the execution stream cannot be completed. It could be due to an error in fetching instructions, loading/storing data, etc.

Asynchronous machine check sources are also maskable by the *ME* bit in the *MSR* register. Typical example: access to nonexisting or corrupted memory.

1.1.3 IVOR2, IVOR3: data storage, instruction storage

Data and instruction storage exceptions are raised if the access control rules are violated. Both CMPU and SMPU can raise a storage exception.

Typical example: access to memory region without permission.

1.1.4 IVOR4: external input

This exception is used for interrupt handling. It is raised if the *p_extint_b* signal is asserted (level sensitive). External input is maskable in the MSR register.

This signal is connected with an interrupt controller (INTC) and is active while there is any interrupt pending. Core can service this exception in autovectored mode or by vector offset, these are usually referred to as software or hardware vector mode.

Typical example: peripheral interrupts

Once external to the core, this includes all interrupt sources from peripheral and platform.

1.1.5 IVOR5, IVOR6: alignment program

Alignment and program exceptions are raised, if the core tries to execute instructions, which are not allowed, nor implemented or if a combination of parameters is invalid for a given instruction.

Alignment exception catches attempt to program word or halfword data to misaligned address. Program exception is raised for invalid access to the SPR and DCR registers and instructions whose opcodes are not implemented.

1.1.6 IVOR7: performance monitor

A performance monitor exception is called if any of the counters reach the overflow condition.

To achieve it, the user has to set up a performance monitor and enable an overflow condition interrupt.

This exception is maskable in MSR (shares mask with external input and/or debug exception).

1.1.7 IVOR8: system call

A system call is a software-triggered exception.

To jump there, the core has to execute the “se_sc” instruction.

1.1.8 IVOR9: debug

Debug exception is raised for various types of event and instruction. The purpose is to interrupt the code execution and have the ability to read or modify core state, register values or to implement any kind of handler to catch and debug erroneous states.

Debug exception can be set to occur on read/write/execute of a specific address, can be called on the execution of a “trap” instruction, branch, return on interrupted service entry.

A complete list of debug events capable of triggering this exception is available in the core documentation.

The debug exception is maskable in the MSR and DBCR0 register.

With standard debug tools (using JTAG) it is not needed to program and use debug exceptions. Debug exception is more likely to be used for specific cases, where it is used to support features such as “Debug-over-CAN”, “Debug-over-Ethernet”, etc.

1.1.9 IVOR10, IVOR11: floating-point data, floating-point round

Floating-point related exceptions are called when the number of computation results in error, or if the result is not exact.

Data exception catch inputs or results of the FPU operation, which are considered to be not a regular number (such as infinity or NaN), operations with such numbers and operations, which could lead to underflow or overflow.

Typical example: divide by zero.

Round exception is not necessarily a reporting error, but likely an unexactness of operation result. In this case, we are getting informed that the output was rounded and we lost certain precision.

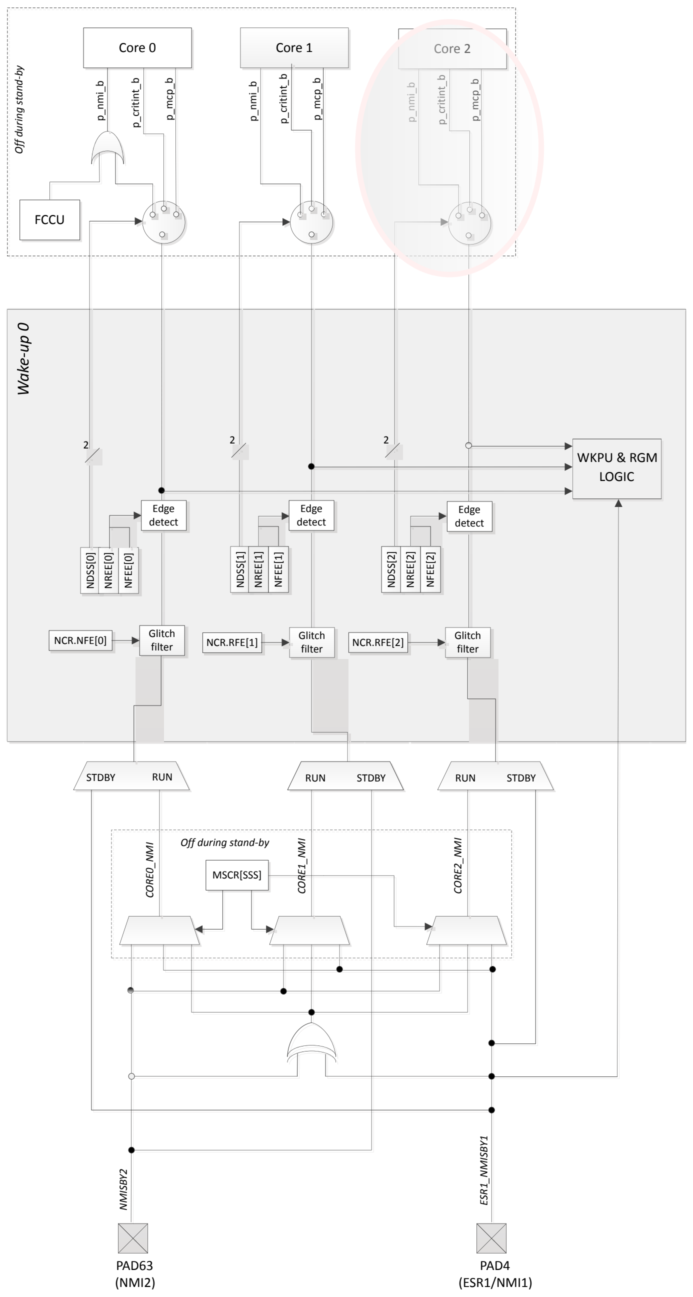
Typical example: result of division 2.0/3.0.

2 Exception triggers

2.1 Critical input, NMI input and machine check request signals

SPC58 devices are capable of routing NMI pad inputs (*NMI1* and *NMI2*) as wakeup requests and/or core interrupts.

Figure 1. NMI to core input routing



For each core we can select among NMI request (IVOR1), machine check request (IVOR1) or critical input (IVOR0).

The selection is made by programming wake-up unit *NCR* register bitfields *NDSSx* and *NREEx* (*NFEEx* respectively) wakeup unit (WKPU) periphery.

Note: *In system integration unit lite2 (SIUL2) periphery an appropriate input SIUL2.MSCR_MUX register and its SSS field has to be set to route pad input to NMI lines.*

2.2 Exception triggering examples

The following examples and references are made with SPC58NH92, it could occur that some indexes or offsets have to be updated for other devices, but in principle they share the same process.

Some exceptions are possible to trigger with various conditions, but only a selection of them is shown.

2.3 IVOR 0

Critical input can be called only by asserting the *p_critint_b* signal. Exception maskable in the *MSR*register, to enable it the *CE* bit has to be set '1'.

Core input is selected by the NDSSx field. Both rising and falling edges can be enabled.

The SIUL module is then responsible for pad selection.

In the example below, *CORE2_NMI*signal is set by *MSCR[812].*

In the signal table refer to *NMISBY2*.

```
/*Enable rising edge and route NMI2 to crit_b of core 2 */ SIUL2.MSCR_MUX[812-512].B.SSS =0x0
2; //selectPD15
WKPU_0.NCR.B.NDSS2 = 0x01; //Critical ISR
WKPU_0.NCR.B.NREE2 =0x01; //Rising edge
```

Note: *Input pad configuration is not shown (IBE, ILS, etc.). In this case, a rising edge on PD[15] will trigger IVOR0.*

2.3.1 Handling of IVOR0

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*).

Critical interrupt follows the “standard” way of exception handling, all (used) registers have to be backup and restored afterwards.

Return address and core status are saved to the *CSRRx* set of registers.

Interrupt status is reported in the *WKPU_0.NSR* register, where the *NIFx* flag must be cleared. Otherwise, the overrun condition may be signaled (*NOVFx*) on the next NMI assertion.

IVOR 0 exception service routine shall be exited by "se_rfci" instruction for correct return.

2.4 IVOR 1

2.4.1 MCP request

Machine check can be called by asserting the *p_mcp_b* signal.

The exception is maskable in the *HID0* register (*SPR1008*), to enable it the *EMCP* bit has to be set '1'. The exception is maskable in the *MSR* register, to enable it the *ME* bit has to be set '1'.

```
WKPU_0.NCR.B.NDSS2 = 0x02; //Machine check request
```

2.4.2 NMI input

Machine check can be called by asserting the *p_nmi_b* signal. This exception source is not maskable.

WKPU_0.NCR.B.NDSS2 = 0x00; //NMI

As shown in the Figure 1. NMI to core input routing the path between NMI and core has to be properly set up. Core input is selected by the NDSSx field.

Both rising and falling edges can be enabled.

The SIUL module is then responsible for pad selection.

In the example below, the *CORE2_NMI* signal is set by *SIUL2.MSCR[812].*

(In signal table refer to *NMISBY2*).

```
/* Enable rising edge and route NMI2 to machine check of core 2 */ SIUL2.MSCR_MUX[812-512].B.
SSS = 0x02; //select PD15
WKPU_0.NCR.B.NDSS2 = NMI or MCP, see above.
WKPU_0.NCR.B.NREE2 = 0x01; //Rising edge
```

Note: Input pad configuration is not shown (IBE, ILS, etc. in SIUL2_MSCR register). In this case, rising edge on PD[15] pad will trigger IVOR1.

2.4.3 Error condition

Machine check is taken also on the following errors: Asynchronous sources (maskable in the core *MSR* register)

- Instruction or data cache parity error
- Instruction or data cache lock error
- ISI or Bus error on the first instruction fetch for an exception handler
- Stack limit check failure
- IMEM or DMEM parity error

Synchronous sources (error reports, not maskable)

- Instruction fetch error
- Load or store type instruction error
- Guarded instruction error
- EDC/ECC error signaled on cache or local memory access

For example:

```
/* Trigger machine check (load type error) */ uint32_t dummy = *(uint32_t*)0x00;
```

2.4.4 Handling of IVOR1

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x10.

Machine check interrupt follows the “standard” way of exception handling, all (used) registers have to be backup and restored afterwards.

Return address and core status are saved to the *MCSRRx* set of registers.

Power PC architecture implements various kinds of exception, which allow to save the core state, react to asynchronous event or failure condition and then restore and continue the previous context.

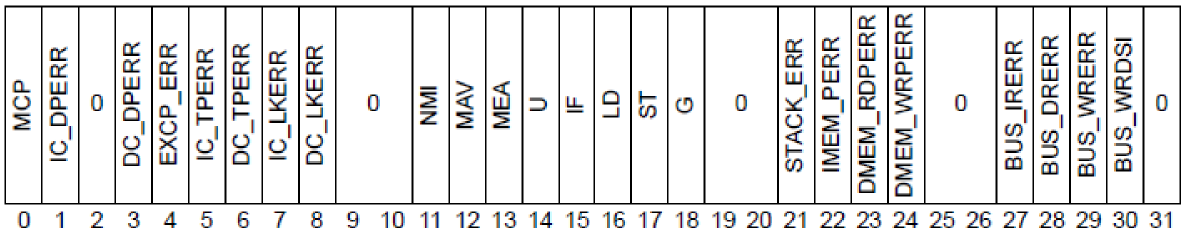
This technical note describes all available core interrupt vectors related to the exceptions and additionally provides a guideline on how to invoke them and how to service them.

The exception mechanism also allows the processor to escalate to supervisor state. This example is also described.

2.4.5 Determining the error source

Machine check handling involves its flag register. The first level of error source determination reads the syndrome register (*MCSR, SPR572*).

Figure 2. **Machine check syndrome register**



Just by reading this register we can differentiate if the exception is taken by an external signal (*NMI*, *MCP*) or if it is an error condition linked to cache, bus etc.

If the error condition is leading to load or store, the next step is to track where the access failed and eventually from which part of the application.

- Failed access location:
  Load and store (including cache fetch) errors report their address in the *MCAR* register.
  The MAV bit in *MCSR* is set to indicate that a valid address is latched.

- Instruction pointer (IP):
  If the handler could take in account which part of the code failed (for example, specific task, driver, etc.) such information is readable natively from the backup-restore register *MCSRR0*.
- Eventually (if used) the PID0 register could be read to determine the process ID

There is no generic rule on how to handle such situations because it is highly application specific and error type dependent. Some error conditions are possible to resolve, some are not recoverable.

Giving the ECC error as an example: if the handler is capable of reinitializing the ECC segment and restoring the data, then we could consider it recoverable and the application may resume. If there are no valid data to use as backup and the application cannot safely continue, we should consider it as not recoverable.

2.4.6 Instruction skipping

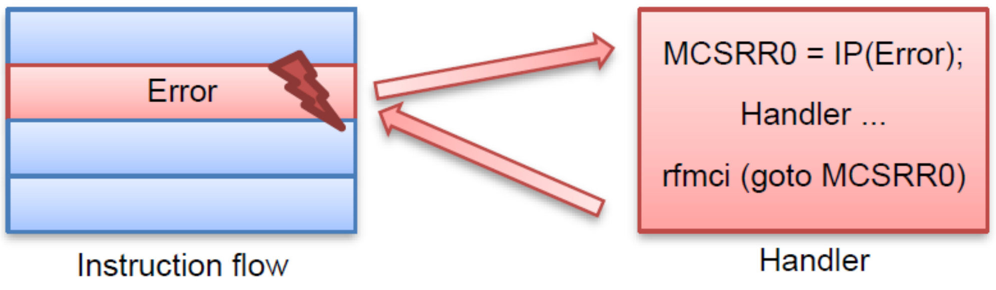
Instruction skipping is useful to unblock the core in case there is no recovery to the error condition (if the impact of a skipped instruction is not fatal to the execution flow).

By the principle of interrupt handling the core should return to the point just before the interrupt was taken and with the same register context. In fact, the core returns to the address kept in the *MCSRR0* register.

Because the instruction pointer was not advanced at failed execution, the *MCSRR0* points exactly to the instruction, which caused the error (it could be useful information, see above).

As a consequence, the failing instruction is reexecuted after the exception exit.

Figure 3. Endless error loop



In order to break such a loop the return address has to be adjusted. If possible, the handler has to read the instruction opcode, determine the size of the instruction (16 or 32 bits) and increment the *MCSRR0 register* accordingly.

Figure 4. **MCSRR0 update based on instruction size detection**

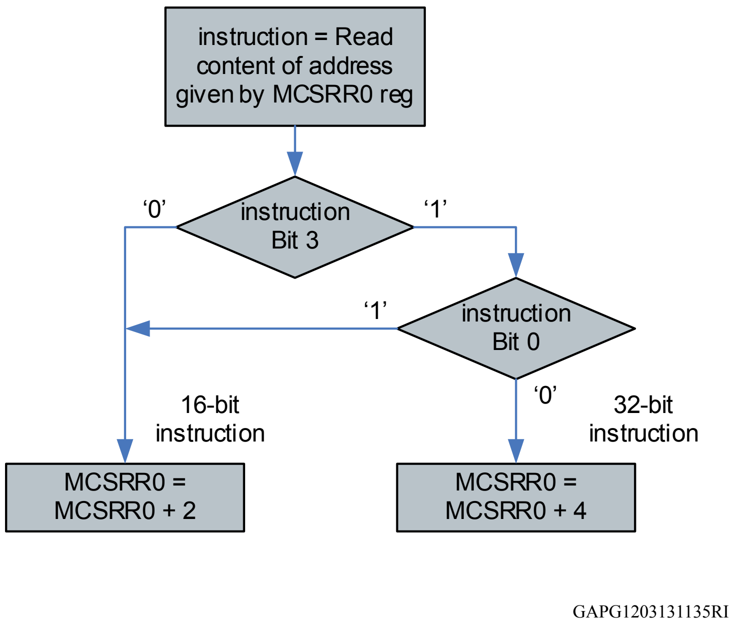
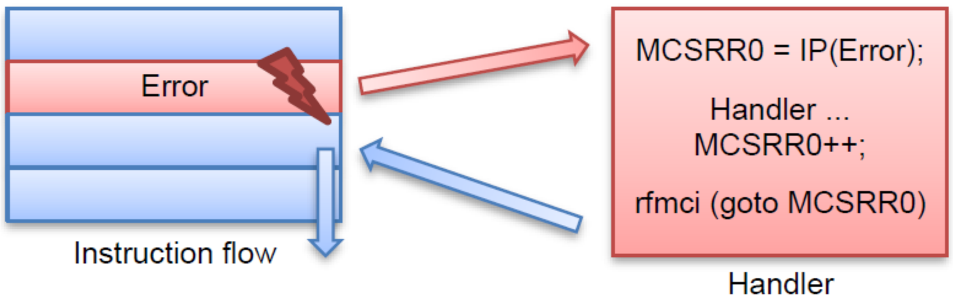


Figure 5. **Instruction skipping by MCSRR0 update**



Compilers may have prebuilt prologues and epilogues, which may not allow them to directly change. *MCSRR0*, check available intrinsics with your compiler vendor.
Exception is released by the "***se_rfmci***" instruction.

2.5 IVOR 2

A data storage exception is raised on the access rules violation when executing a load or store instruction. Both memory protection units (SMPU, CMPU) can restrict access to a given region and cause this exception.
All accesses blocked by SMPU

```
//Create SMPU restricted region
_smpu->RGD[1].WORD0.R = SMPU_REGION_START; //set region address
_smpu->RGD[1].WORD1.R = SMPU_REGION_END;
_smpu->RGD[1].WORD2_FMT0.R = 0x00000000; //no permissions
_smpu->RGD[1].WORD3.B.VLD = 1; //validate region uint32_t dummy = *(uint32_t*)SMPU_REGION_STA
RT; //trigger exception
```

A detailed description of SMPU and CMPU is not in the scope of this note.

2.5.1 Handling of IVOR2

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x20. IVOR2 uses a common set of save-restore registers the *SRRx*, standard prologue, and epilogue can be used.

Reaction to the data storage exception is purely application specific. To determine which module violated the rules and which memory was the target, the user can read the various registers.

- PID0 (SPR48): process ID register, read which process caused an exception.
- ESR (SPR62): exception syndrome, determine read or write type of access.
- DEAR (SPR61): read which address reported the exception.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction.

In the case of SMPU, more flags are readable directly from the peripheral.

2.5.1.1 Instruction skipping

Similarly to HW-related bus errors, we can skip a failing instruction. See skipping instruction described in Section 1: General introduction.

Exception is released by the “***se_rfi”*** instruction.

2.6 IVOR 3

Instruction storage exception is raised on access rules violation during instruction fetch.

Both memory protection units (SMPU, CMPU) can restrict access to a given region and cause this exception.

- All accesses blocked by SMPU

```
//Create SMPU restricted region
_smpu->RGD[1].WORD0.R = SMPU_REGION_START; //set region address
_smpu->RGD[1].WORD1.R = SMPU_REGION_END;
_smpu->RGD[1].WORD2_FMT0.R = 0x00000000; //no permissions
_smpu->RGD[1].WORD3.B.VLD = 1; //validate region
((void(*)(void))SMPU_REGION_START)(); //trigger exception
```

A detailed description of SMPU and CMPU is not in the scope of this note.

2.6.1 Handling of IVOR3

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x30. IVOR3 uses a common set of save-restore registers *SRRx*,

A standard prologue and epilogue can be used.

Reaction to instruction storage exception is purely application specific. To determine which module violated the rules and which memory was the target, the user can read the various registers.

- PID0 (SPR48): process ID register, read which process caused an exception.
- ESR (SPR62): exception syndrome, determine read or write type of access.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction. In the case of SMPU, more flags are readable directly from the peripheral.

Exception is released by the **“*se_rfi”*** instruction.

2.7 IVOR 4

External interrupt is taken on assertion of *p_extint_b* signal, which is driven by platform interrupt controller *INTC*.

External interrupt is maskable in the *MSR* register, to enable it the user has to set the *EE* bit to ‘1’.

It is the most common interrupt used for all the peripherals to trigger asynchronous events. The interrupt controller is responsible for arbitration between events based on the priority scheme and propagate the top-most interrupt event in the *IACKR* register and assert the *p_extint_b*.

For example, software triggered interrupt

```
//Trigger software interrupt 0
INTC_1.SSCIR[0].B.SET = 0x01;
```

Note: A detailed description of the INTC_1 platform interrupt controller is not in a scope of this document. See reference manual [1] for more details.

2.7.1 Handling of IVOR4

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x40.

Core receives an interrupt and has to read the event from *IACKR*. This will also acknowledge that an interrupt is taken and if there is no other pending interrupt the *p_extint_b* line is deasserted.

After the interrupt is serviced, the core should write the *EOIR* register to signal the end of the interrupt routine to the controller. This restores INTC priority, which was latched before the interrupt was taken.

A complete description of the interrupt controller, its vector modes and core-to-*INTC* handshake are not in the scope of this note. Check the reference manual for details (refer to [1]).

2.7.2 Preemption

The interrupt controller is capable of preemption based on the event priority. If there is a pending event with higher priority than the current service interrupt, the request is asserted and the core (if *MSR.EE* is reenabled) can receive another IVOR4 exception.

Exception is released by the "***se_rfi***" instruction.

2.8 IVOR 5

Alignment exception is raised if the address is not properly aligned to the respective data size. It will occur in the following cases:

- Word access is not aligned to 32 bits (*lwarx, stwcx*, etc.)
- Multiword access is not aligned to 32 bits (*stmw, lmw*)
- Halfword access is not aligned to 16 bits (*lharx, sthcx*, etc.)
- Data cache block set to zero (*dcbz*) instruction is attempted
- LSP/SPE access is not properly aligned

For example:

```
/* Misaligned multiword write */
asm("e_addi r7,r1,0x8"); /* Address word in stack */
asm("e_addi r3,r7,0x1"); /* Shift address by one byte */
asm("e_stmw r30,0x0(r3)"); /* Store to misaligned address*/
```

2.8.1 Handling of IVOR5

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x50. IVOR5 uses a common set of save-restore registers *SRRx*,

A standard prologue and epilogue can be used.

Reaction to alignment exception is purely application-specific. To determine which module caused the error and which memory was the target, the user can read the various registers:

- PID0 (SPR48): process ID register, read which process caused an exception.
- ESR (SPR62): exception syndrome, determine read, or write type of access.
- DEAR (SPR61): read which address reported the exception.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction. Exception is released by the **"*se_rfi*"** instruction.

2.9 IVOR 6

Program exception is called if the core tries to execute instructions, which are illegal, not implemented or if the core access level is not sufficient (user mode).

They occur in the following cases:

- Instruction is not implemented
- Instruction is illegal (*lswi, lswx, stswi, stswx, mfapidi, mfdcrx, mtdcrx*)
- Instruction is illegal (PowerISA 2.06 floating-point instruction)
- Access to SPR or DCR registers, which does not exist
- Instruction is privileged but the core state (*MSR.PR*) is set to the user
- Trap instruction while debug is disabled

For example:

```
/* Execute invalid instruction */
asm("mfapidir0,r0");
```

2.9.1 Handling of IVOR6

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x60. IVOR6 uses a common set of save-restore registers *SRRx*,

A standard prologue and epilogue can be used.

Reaction to a program exception is application specific.

It could be possible to resolve privileged access, but there are very limited options for recovering instructions, which are not implemented.

To determine the character of the failure, the user shall read the *ESR* register (*SPR62*) and check flags *PIL, PPR, PTR, and* eventually *FP*.

Other registers could be used to read more about which module caused the exception.

- PID0 (SPR48): process ID register, read which process caused an exception.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction. Exception is released by the **“*se_rfi*** ” instruction.

2.10 IVOR 7

A performance monitor interrupt is asserted if any of its internal counters reaches a given threshold. It is internally referred to as "overflow condition". Such interrupt has to be enabled in the *to PMLCa* register for each channel.

Performance monitor interrupt shares a mask with an external input (*MSR.EE*) or debug exception (*MSR.DE*).

Special care has to be taken when to the debugger is attached to the device. Usually, resources of performance monitor are allocated the debugger and the core may or may not be able to use them.

For example, generate a performance monitor interrupt after one processor cycle.

```
//PMGC0 (reg400)
asm("e_lis r3, 0x4000"); //Set PMIE = 1 ~ enable IRQ
asm("mtpmr400,r3");
```

```
//PMLCa0 (reg144)
asm("e_lis r3, 0x0401"); //Enable overflow condition to generate IRQ,
asm("mtpmr144,r3"); // selected event = processor cycles
```

```
//PMLCb0 (reg272)
asm("e_li r3, 0x2000"); //Set PMC0 as triggered
asm("mtpmr272, r3");
```

```
//PMC0 (reg16)
asm("e_lis r3, 0x7FFF");
asm("e_or2i r3, 0xFFFF");
asm("mtpmr16, r3"); /* Wait 1 tick */
```

2.10.1 Handling of IVOR7

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x70. IVOR7 uses a common set of save-restore registers *SRRx*,

A standard prologue and epilogue can be used.

Performance monitor interrupt could be used to finalize or evaluate performance measurement, to restart it or to serve for any other purpose linked to the performance monitor.

Exception is released by the **“*se_rfi”*** instruction.

2.11 IVOR 8

System call is a sort of branch that the core can execute. The purpose and functionality implemented in a system call handler are application specific.

```
//Execute system call
asm("se_sc");
```

It could be understood as a special case of branch instruction, where the address is fixed, but we are entering it with the privileged access (*MSR.PR* = 0).

System call could be used to change mode from limited user mode to privileged supervisor mode practically. SRR0 register content (contains saved MSR register) could be modified (MSR.PR bit) inside IVOR 8 exception service routine before released by se_rfi. Updated MSR content would be restored from SRR0.

2.11.1 Handling of IVOR8

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x80. If required a standard prologue and epilogue can be used.

The system call is released by the **"*se_rfi*"** instruction.

2.12 IVOR 9

Debug exception is raised for various debug events (if enabled). There are multiple sources for them and the user has to read the DBSR register to differentiate which one caused it.

Debug interrupt can be enabled for

- Instruction complete
- Branch taken
- Interrupt taken
- Critical interrupt taken
- Trap instruction
- Instruction address compares
- Data address compares
- Debug notify interrupt
- Interrupt return
- Critical interrupt return
- External debug
- Performance monitor
- MPU
- Imprecise debug event

To receive debug exception after the condition is met, it has to be programmed in the DBCR0 register.

Also be aware that debug resources are shared with hardware (debugger) and may interfere with it.

- Enable ICMP (instruction complete) interrupt.

```
asm("e_lis r7,0xC8000000");
asm("mtspr spr308, r7"); //Program DBCR0 - ICMP
asm("nop"); /* Execute instruction to trigger ICMP */
```

2.12.1 Handling of IVOR9

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0x90. Return address and core status are saved to the *DSRRx* set of registers.

The core implements a lot of debug functionalities, which are not in the direct focus of this note, just as example we are listing registers where the status can be read and address registers associated with compare events.

- DBSR(SPR304): debug status and flags
- DDEAR(SPR600): debug effective address (from compare event)
  Exception is released by the **"*se_rfdi*"** instruction.

2.13 IVOR 10

This exception is taken if EFPU (embedded floating-point unit) input or output data are invalid. This includes the following cases.

- The operand of instruction is infinity, NaN, denorm or zero-by-zero division. (*FINV*)
- Division by zero (*FDBZ*)
- Overflow or underflow (*FOVF, FUNF*)

The exception for each operation is masked in the *EFPU_CTR* register (*SPR512*).

for example:

```
/* EFPU invalid operation */
MTSPR(512, 0x20); //Enable FINVE
float A = 0.0;
float B = 0.0;
B = A/B; //Divide zero by zero
```

2.13.1 Handling of IVOR10

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0xA0. IVOR10 uses a common set of save-restore registers *SRRx*, a standard prologue and epilogue can be used.

To differentiate between the above cases, the user has to read the *SPEFSCR* register.

Reaction to the EFPU exception is application specific. To determine which module caused the error the user can read some various registers.

- PID0 (SPR48): process ID register, read which process caused an exception.
- ESR (SPR62): exception syndrome.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction.

2.13.2 Instruction skipping

Similarly to HW-related bus errors, we can skip a failing instruction. See the skipping instruction described in Section 1: General introduction.

Exception is released by the **“*se_rfi*”** instruction.

2.14 IVOR 11

This exception is taken if EFPU (embedded floating-point unit) output data are imprecise. This includes the following cases:

- The result of operation is not exact
- Overflow or underflow occurs while data exception is disabled

If the floating-point data exception occurs, it has precedence over the round exception. The rounding exception is masked in the *EFPU_CTR* register (*SPR512*).

For example:

```
/* EFPU inexact operation */
MTSPR(SPR_EFPU_CTR, 0x40); //Enable FINXE
float A = 2.0; float B = 3.0; B = A/B;
```

2.14.1 Handling of IVOR11

Interrupt is taken on the address programmed to the *IVPR* register (*SPR63*) and offset 0xB0. IVOR11 uses a common set of save-restore registers *SRRx*, a standard prologue, and epilogue can be used.

To differentiate between the above cases, the user has to read the *SPEFSCR* register.

Reaction to the EFPU exception is application specific. To determine which module caused the error the user can read various registers.

- PID0 (SPR48): process ID register, read which process caused an exception.
- ESR (SPR62): exception syndrome.
- SRR0 (SPR26): generic save-restore register, read address of failing instruction. Exception is released by the **“*se_rfi*”** instruction.

Appendix A Reference documents

[1] Reference manual *SPC58 H Line - 32 bit Power Architecture automotive MCU Triple z4 cores 200 MHz, 10 MBytes Flash, HSM, ASIL-D* (RM0452)

[2] Reference manual *e200z0 and e200z0h CPU cores* (RM0044).

Revision history

Table 1. Document revision history

Date	Revision	Changes
02-Dec-2025	1	Initial release.

Contents

1 **General introduction . . . 2**
1.1 IVORs list . . . 2
1.1.1 IVOR0: critical input . . . 2
1.1.2 IVOR1: machine check . . . 2
1.1.3 IVOR2, IVOR3: data storage, instruction storage . . . 2
1.1.4 IVOR4: external input . . . 2
1.1.5 IVOR5, IVOR6: alignment program . . . 2
1.1.6 IVOR7: performance monitor . . . 2
1.1.7 IVOR8: system call . . . 3
1.1.8 IVOR9: debug . . . 3
1.1.9 IVOR10, IVOR11: floating-point data, floating-point round . . . 3
2 **Exception triggers . . . 4**
2.1 Critical input, NMI input and machine check request signals . . . 4
2.2 Exception triggering examples . . . 5
2.3 IVOR 0 . . . 5
2.3.1 Handling of IVOR0 . . . 5
2.4 IVOR 1 . . . 5
2.4.1 MCP request . . . 5
2.4.2 NMI input . . . 5
2.4.3 Error condition . . . 6
2.4.4 Handling of IVOR1 . . . 6
2.4.5 Determining the error source . . . 6
2.4.6 Instruction skipping . . . 7
2.5 IVOR 2 . . . 8
2.5.1 Handling of IVOR2 . . . 9
2.6 IVOR 3 . . . 9
2.6.1 Handling of IVOR3 . . . 9
2.7 IVOR 4 . . . 9
2.7.1 Handling of IVOR4 . . . 10
2.7.2 Preemption . . . 10
2.8 IVOR 5 . . . 10
2.8.1 Handling of IVOR5 . . . 10
2.9 IVOR 6 . . . 10
2.9.1 Handling of IVOR6 . . . 11
2.10 IVOR 7 . . . 11
2.10.1 Handling of IVOR7 . . . 11

2.11 IVOR 8 11
2.11.1 Handling of IVOR8 12
2.12 IVOR 9 12
2.12.1 Handling of IVOR9 12
2.13 IVOR 10 12
2.13.1 Handling of IVOR10 13
2.13.2 Instruction skipping 13
2.14 IVOR 11 13
2.14.1 Handling of IVOR11 13
Appendix A Reference documents 14
Revision history 15