EPPC Exception Processing

Exception Terms

User Mode	The Privilege Level that Applications run in.
Supervisor Mode	The Privilege Level that the Operating System runs in. Also called “Privileged Mode”
Exception	An event which causes deviation from normal processing. Examples: - Interrupt (internal or external) - Resets - Bus error
Ordered Exception	No program state is lost after the exception (the machine state is saved).
Unordered Exception	Program state may be lost after the exception. Includes reset, machine check and other non-maskable exceptions.
Asynchronous Exception	Exception not caused by an instruction.
Synchronous Exception	Exception caused by an instruction.
Precise Exception	The exact processor context when the exception occurred is available, and the exact cause of the exception is always known. – Processor backs the machine up to the instruction which caused the exception
Imprecise Exception	The exact processor context is not known when the exception is processed, because concurrent operations have affected the information that comprises the processor context.
Maskable Exception	May be masked by the operating system.

Exception Classes

CLASS	EXCEPTION TYPE
ASYNCHRONOUS, UNORDERED	RESET, NON-MASKABLE
SYNCHRONOUS, UNORDERED	MACHINE CHECK (BUS ERROR)
ASYNCHRONOUS, ORDERED	EXTERNAL INTERRUPT DECREMENTER, PIT INTERRUPTS
SYNCHRONOUS (ORDERED, PRECISE)	INSTRUCTION -CAUSED EXCEPTIONS

ORDERED EXCEPTIONS - When the Exception is taken, No program state is lost.

UNORDERED EXCEPTIONS - When the Exception is taken, the program state is unrecoverable.

> ***Reset*** and ***Machine Check*** Exceptions are unrecoverable, if they occur during the servicing of another exception.

PRECISE EXCEPTIONS - When the exception is taken, the processor backs the machine up to the instruction causing the exception. The instruction causing the exception may not have begun execution, may partially be completed, or may have completed execution.

> The Core implements all storage associated interrupts as precise interrupts. This means that a load/store instruction is not complete until all possible error indications have been sampled from the Load/Store Bus.

Exception Definitions (1 of 2)

System Reset Interrupt	Occurs when the NMI pin is asserted, SWT times out, Hard or Soft Reset pins are asserted
Machine Check Interrupt	The accessed address does not exist or a data error was detected
Data Storage Interrupt	Never generated by the hardware The software may branch to this location as a result of either Implementation Specific Data TLB error interrupt or Implementation Specific Data TLB miss int.
Instruction Storage Interrupt	Never generated by the hardware The software may branch to this location as a result of an Implementation Specific Instruction TLB error interrupt
Alignment Interrupt	**Occurs as a result of one of the following cases:** The operand of a Floating-Point load or store is not word aligned. The operand of Load/Store multiple is not word aligned. The operand of lwarx or stwcx. is not word aligned. The operand of Load/Store individual scalar instruction is not naturally aligned when MSR_{LE} = 1. An attempt to execute multiple/string instruction is made when MSR_{LE} = 1
Program Interrupt	**Floating-Point Enabled Exception** type Program interrupt is not generated by the EPPC. **Illegal Instruction** type program interrupt is not generated by the Core, an Implementation Dependent Software Emulation Interrupt is generated instead **Privileged instruction** type Program interrupt is generated for on core valid SPR field or any SPR encoded as an external to the core special register if spr_0=1 and MSR_{PR}=1, as well as an attempt to execute privileged instruction when MSR_{PR}=1.
Floating Point Unavailable Interrupt	Not generated by the EPPC An Implementation Dependent Software Emulation Interrupt will be taken on any attempt to execute Floating-Point instruction regardless of MSR_{FP}

Exception Definitions (2 of 2)

Trace Interrupt	Occurs If MSR_{SE} = 1 and any instruction except rfi is successfully completed, or MSR_{BE} = 1 and a branch is completed
Floating Point Assist Interrupt	Not generated by the EPPC An Implementation Dependent Software Emulation Interrupt will be taken on any attempt to execute Floating-Point instruction
Implementation Dependent Software Emulation Interrupt	**Occurs as a result of one of the following cases:** •When there is an attempt to execute any non implemented instruction. (This include all Illegal and unimplemented optional instructions and all floating point instructions). •When there is an attempt to execute a mtspr or mfspr which specifies on core non implemented register. (regardless of spr_0). •When there is an attempt to execute a mtspr or mfspr which specifies off core non implemented register and spr_0=0 or MSR_{PR}=0
Implementation Specific Instruction TLB Miss Interrupt	Occurs when MSR_{IR}=1 and there is an attempt to fetch an instruction from a page that its Effective Page Number can not be translated by the Instruction TLB
Implementation Specific Instruction TLB Error Interrupt	**Occurs in the following cases:** •The effective address cannot be translated (either Segment valid bit or Page valid bit of this page are cleared in the translation table) •The fetch access violates storage protection •The fetch access is to Guarded storage and MSR_{IR}=1
Implementation Specific Data TLB Miss Interrupt	Occurs when MSR_{DR}=1 and there is an attempt to access a page that its Effective Page Number can not be translated by the Data TLB
Implementation Specific Data TLB Error Interrupt	**Occurs in the following cases:** •The effective address of a Load, Store, icbi, dcbz, dcbst, dcbf or dcbi instruction cannot be translated (either Segment valid bit or Page valid bit of this page are cleared in the translation table) •The access violates the storage protection •An attempt to write to a page with negated Change Bit

Interrupt Priority Mapping

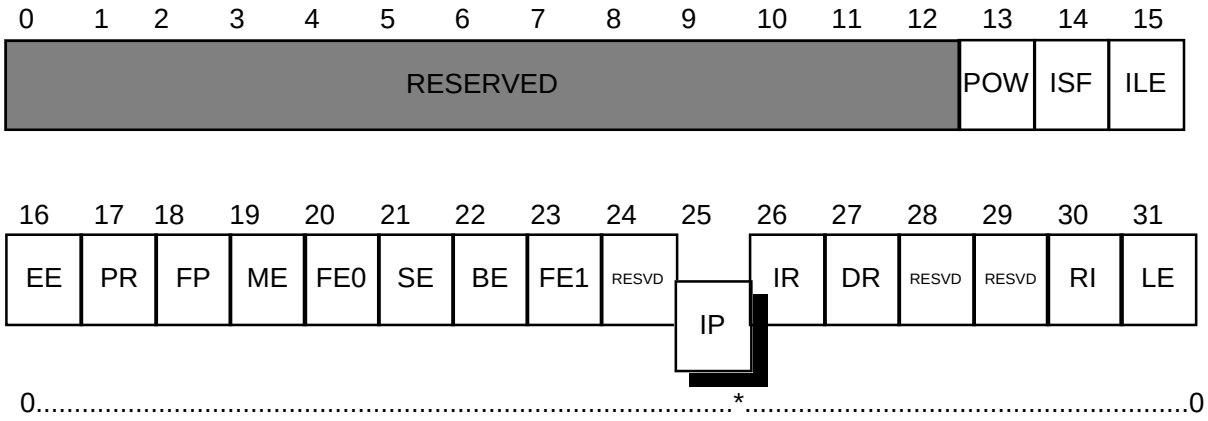
#	Interrupt Type	Caused By
#1	Development non-maskable interrupt	Signal from the Development Port
#2	System reset	NMI_L assertion
#3	Instruction Related Interrupts	Instruction Processing
#4	Peripheral breakpoint request or Dev Port maskable interrupt	Breakpoint signal from any peripheral
#5	External Interrupt	Signal from the interrupt controller
#6	Decrementer interrupt	Decrementer request

Instruction Related Interrupt Detection Order

#	Interrupt Type	Caused by
#1	Trace	Trace bit asserted
#2	Implementation Dependent Instruction TLB miss	Instruction MMU TLB Miss
#3	Implementation Dependent Instruction TLB error	Instruction MMU protection/translation error
#4	Machine Check Interrupt	Fetch Error
#5	Debug I- Breakpoint	Match detection
#6	Implementation Dependent Software Emulation Interrupt	Attempt to invoke un-implemented feature
#7	Floating-Point Unavailable	Attempt is made to execute Floating-Point instruction and MSRFP=0
#8	Privileged Instruction	Attempt to execute priviledged instruction in problem mode
	Alignment Interrupt	Load store checking
	System Call Interrupt	SC Instruction
	Trap	Trap Instruction
#9	Implementation Dependent Data TLB miss	Data MMU TLB Miss
#10	Implementation Dependent Data TLB error	Dat MMU TLB Protection/translation error
#11	Machine Check Interrupt	Load or store access error
#12	Debug -L Breakpoint	Match detection

Vector Table

MSR - MACHINE STATE REGISTER

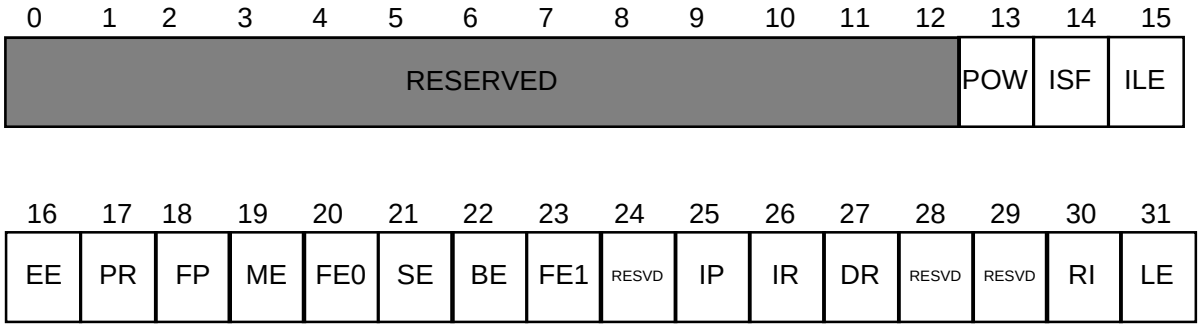


IP = 0 Vector Table Address at 0x00000000
= 1 Vector Table Address at 0xFFF00000.

VECTOR OFFSET (HEX)	EXCEPTION TYPE	
0 0000	RESERVED	
0 0100	SYSTEM RESET	HARD & SRESETS
0 0200	MACHINE CHECK	$\overline{\text{TEA}}$ (BUS ERROR)
0 0300	DATA STORAGE	
0 0400	INSTRUCTION STORAGE	
0 0500	EXTERNAL INTERRUPT	$\overline{\text{IRQ}}$[1 :7], PIT, TB, RTC, PCMCIA, CPM
0 0600	ALIGNMENT	ALIGNMENT ERROR
0 0700	PROGRAM	INSTR. TRAPS,ERRORS, ILLEGAL, PRIVILEGED MSR[FP]=0 & F.P. INSTRUCTION ENCOUNTERED
0 0800	FLOATING-POINT UNAVAILIABLE	
0 0900	DECREMENTER	DECREMENTER REGISTER
0 0A00	RESERVED	
0 0B00	RESERVED	
0 0C00	SYSTEM CALL	'SC' INSTRUCTION
0 0D00	TRACE*	SINGLE-STEP OR BRANCH TRACING
0 0E00	FLOATING-POINT ASSIST*	SOFTWARE ASSIST FOR INFREQUENT & COMPLEX FP OPERATIONS
0 1000	IMPLEMENTATION DEPENDENT SOFTWARE EMULATION	
0 1100	IMPLEMENTATION DEPENDENT INSTRUCTION TLB MISS	
0 1200	IMPLEMENTATION DEPENDENT DATA TLB MISS	
0 1300	IMPLEMENTATION DEPENDENT INSTRUCTION TLB ERROR	
0 1400	IMPLEMENTATION DEPENDENT DATA TLB ERROR	
0 1500 - 01BFF	RESERVED	
0 1C00	IMPLEMENTATION DEPENDENT DATA BREAKPOINT	
0 1D00	IMPLEMENTATION DEPENDENT INSTRUCTION BREAKPOINT	
0 1E00	IMPLEMENTATION DEPENDENT PERIPHERAL BREAKPOINT	
0 1F00	IMPLEMENTATION DEPENDENT NON MASKABLE DEVELOPMENT PORT	

MSR After Hard Reset

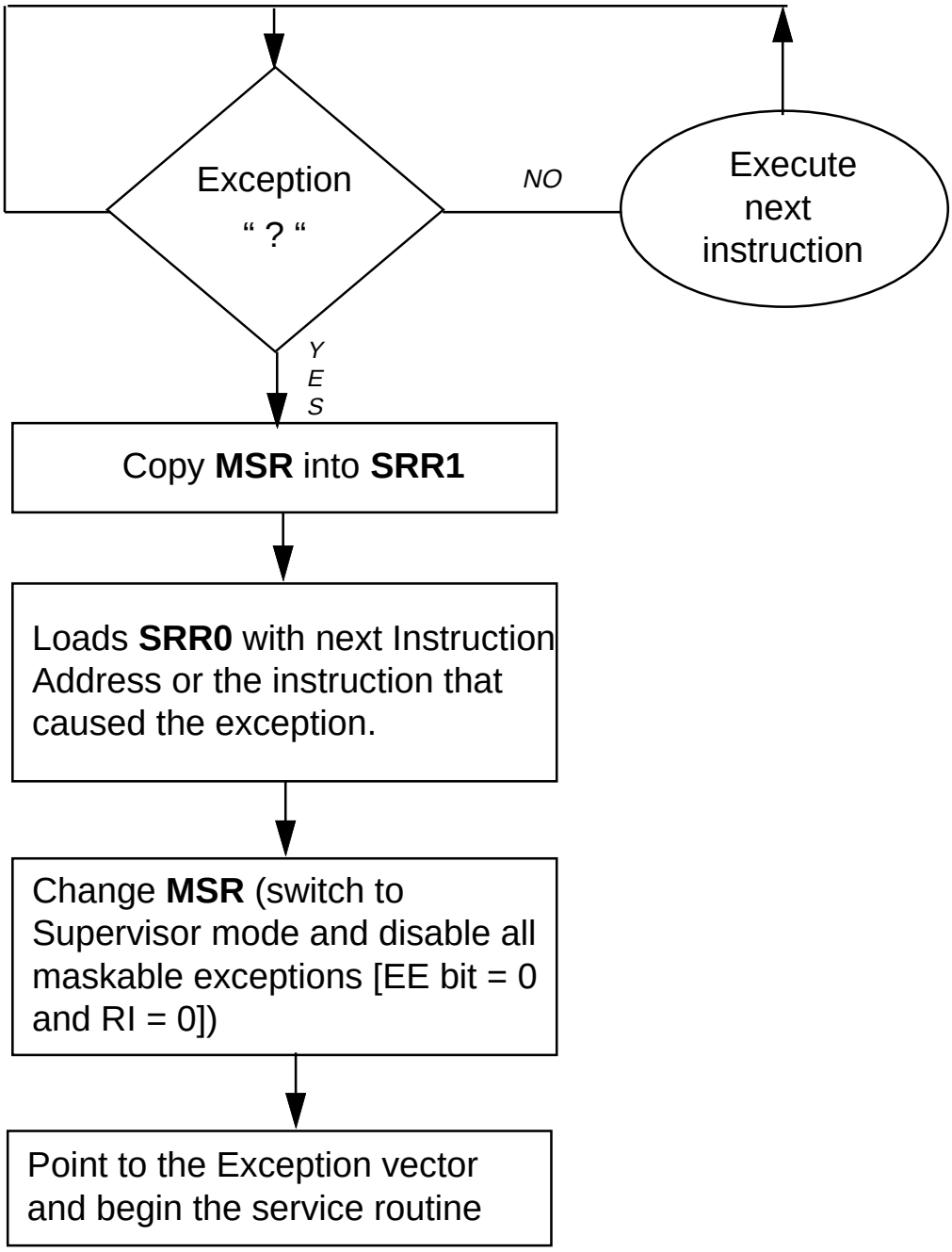
MSR - MACHINE STATE REGISTER



MSR AFTER RESET:

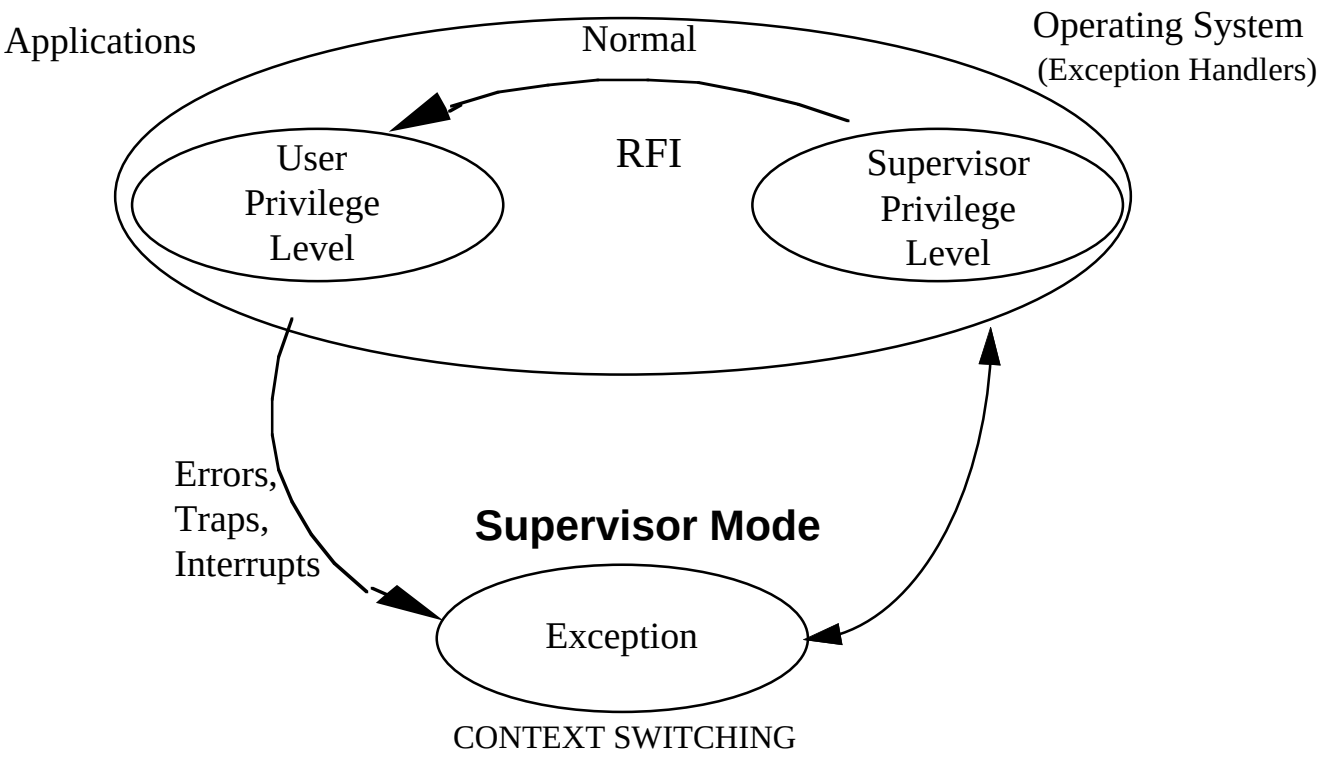
POW	0	Power Management Disable
ISF	0	Implementation Specific Function
ILE	0	Interrupt Little Endian Mode Disabled
EE	**0**	**External and DEC Interrupt are disabled**
PR	0	Privilege Level is Supervisor.
FP	0	Floating Point Unit not available
ME	0	Machine Check Disabled: If transfer error acknowledge (***TEA***) occurs, the Chip will go to Checkstop State. The SIU may assert ***reset*** in order to recover.
FE0	0	Floating-Point Exception Mode 0(has no effect).
SE	0	Single Step Trace Disabled.
BE	0	Branch Trace Disabled.
FE1	0	Floating-Point Exception Mode 1(has no effect).
IP	*****	**Interrupt Prefix . Vector Table Located** at 0x000n - nnnn or at 0xFFFn - nnnn for a value of a “0” or a “1” respectively.
IR	0	Instruction Relocate
DR	0	Data Relocate
RI	**0**	**Recoverable Interrupt Mode is Disabled.**
LE	0	Normal Processing is set for Big Endian Mode.

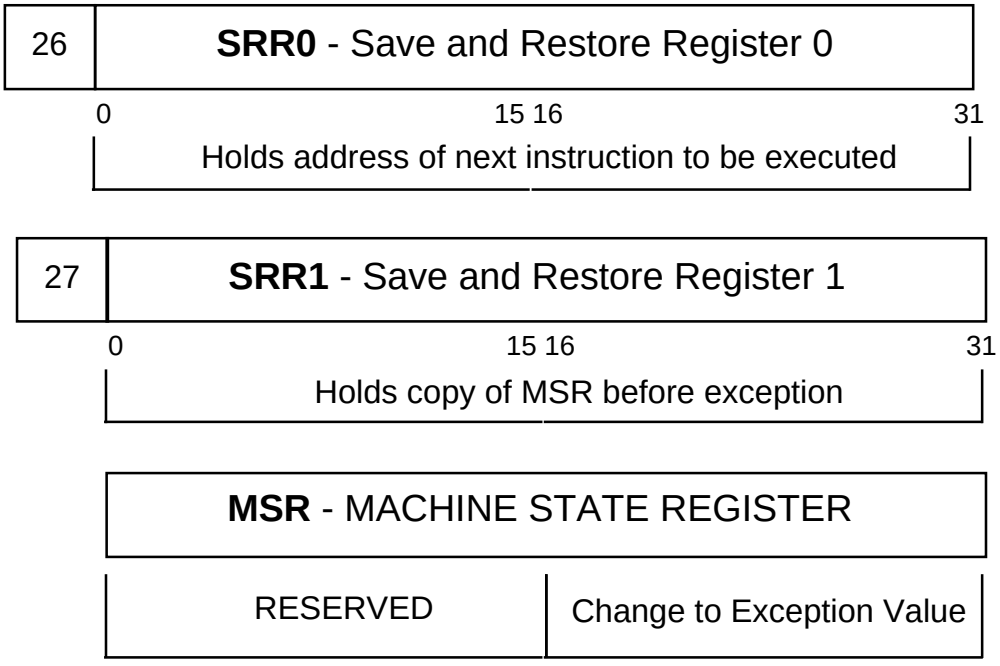
Exception Processing Sequence (1 of 2)



- For most exceptions, the machine state is saved only in **SRR0** and **SRR1.**
- Some exceptions will save other information in **DSISR** and **DAR**:
  - **DSISR (Source Instruction Service Register) -** 7 bit field identifies which instruction caused the exception.
  - **DAR (Data Address Register):** Contains the effective address o the load or store for misaligned exceptions.

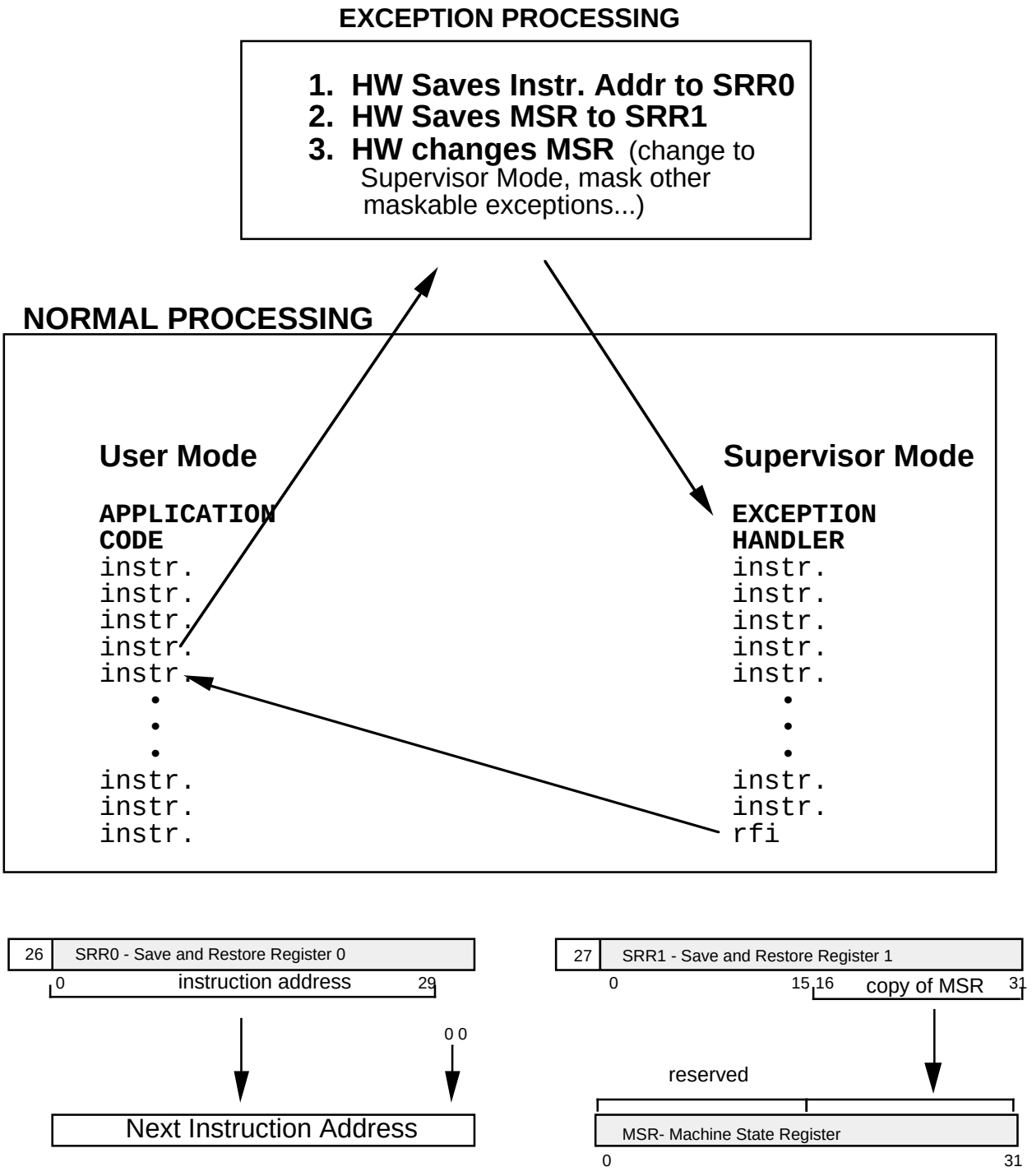
Exception Processing Sequence (2 of 2)





- **SRR0**, **SRR1** and **MSR** are changed after every exception
- All exceptions cause the **core** to enter the supervisor mode.
- The **RFI** instruction restores the Machine State back to User Mode.
- The **RFI** instruction is usually the last instruction in the exception handler.

How the RFI Instruction Operates



RFI Instruction is Supervisor-only used to restore previous Machine State.

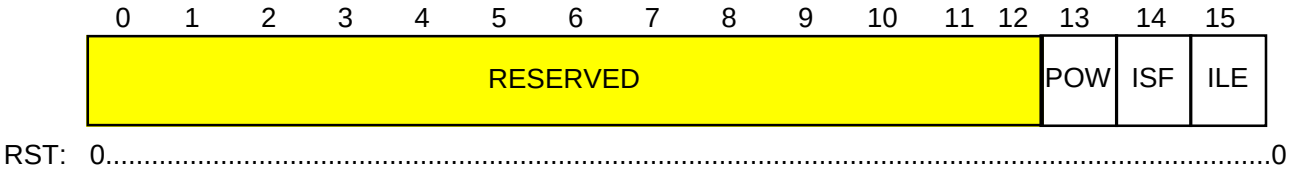
How to Make the ESR Recoverable

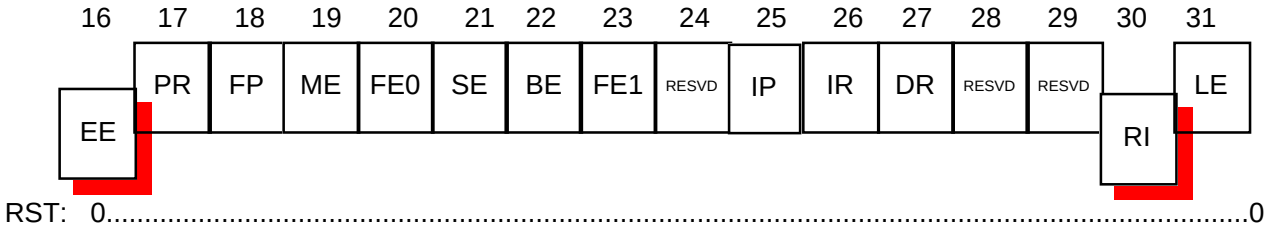
Making the ESR Recoverable

```
asm (“ stwu r9,-12(r1);          /* SAVE R9                */
asm (" mfspr r9,26");            /* PUSH SRR0 ONTO STACK */
asm (" stw r9,4(r1)");
asm (" mfspr r9,27");            /* PUSH SRR1 ONTO STACK */
asm (" stw r9,8(r1)");
asm (" mtspr 80,0");           /* ENABLE INTERRUPTS      */
```

Before ESR Exit

```
asm (" mtspr 82,0");           /* MAKE NON-RECOVERABLE   */
asm (" lwz r9,8(r1)");         /* PULL SRR1 FROM STACK   */
asm (" mtspr 27,r9");
asm (" lwz r9,4(r1)");         /* PULL SRR0 FROM STACK   */
asm (" mtspr 26,r9");
asm (“ lwz r9,0(r1)”);         /* PULL R9 FROM STACK     */
asm (" addi r1,r1,12");
```





Mnemonic	MSR_{EE}	MSR_{RI}	Used For
EIE (80)	1	1	External Interrupt Enable
EID (81)	0	1	External Interrupt Disable, but other interrupts are recoverable
NRI(82)	0	0	Non-Recoverable Interrupt

MOTOROLA
Motorola Technical Training - **MPC860 Course**
Phoenix, Arizona

Title: **ex1.c**
Handling a System Call Exception

Creation Date: **Jan. 10, 1996** From: **68360 Course**

Author: **Bob Bratt**

Description:

The results of this routine are:
1. Initializes the exception vector area with a service routine to increment an LED counter each time a system call instruction is executed.
2. The exception service routine is made recoverable.
Assumptions:
1. Reset conditions exist.

Objective:

If the program executes properly, the LED counter has a count of 1.

Equipment:

MPC860ADS board and UDLP1.

UDLP1 Switch Settings: **N/A**

Connections: **MPC860ADS board and UDLP1 are connected at P13.**

Updates:

ex1.c (1 of 2)

```
/* (EX1.C)                                                    */
#include "mpc860.h"                 /* DUAL PORT RAM EQUATES*/
struct dprbase *pdpr;               /* PNTR TO DUAL PORT RAM*/

main()
{
   void esr();                      /* EXCEPTION SERVICE RTN */
   int *ptrs,*ptrd;                 /* SOURCE & DEST POINTERS*/

   pdpr = (struct dprbase *) (getimmr() & 0xFFFF0000);
                                    /* INIT PNTR TO DPRBASE  */
   ptrs = (int *) esr;              /* INIT SOURCE POINTER   */
   ptrd = (int *)(getevt() + 0xC00); /* INIT DEST POINTER    */
   do                               /* MOVE ESR TO EVT       */
      *ptrd++ = *ptrs;              /* MOVE UNTIL            */
   while (*ptrs++ != 0x4c000064);   /* RFI INTRUCTION        */
   pdpr->PDDAT = 0;                 /* CLEAR PORT D DATA REG */
   pdpr->PDDIR = 0xff;              /* MAKE PORT D8-15 OUTPUT*/
   asm(" sc");                      /* SYSTEM CALL           */
}

#pragma interrupt esr
void esr()
{
   asm (" stwu r9,-12(r1)");       /* PUSH GPR9 ONTO STACK   */
   asm (" mfspr r9,26");           /* PUSH SRR0 ONTO STACK   */
   asm (" stw r9,4(r1)");
   asm (" mfspr r9,27");           /* PUSH SRR1 ONTO STACK   */
   asm (" stw r9,8(r1)");
   asm (" mtspr 80,0");            /* ENABLE INTERRUPTS      */
   pdpr->PDDAT += 1;
   asm (" mtspr 82,0");            /* MAKE NON-RECOVERABLE   */
   asm (" lwz r9,8(r1)");          /* PULL SRR1 FROM STACK   */
   asm (" mtspr 27,r9");
   asm (" lwz r9,4(r1)");          /* PULL SRR0 FROM STACK   */
   asm (" mtspr 26,r9");
   asm (" lwz r9,0(r1)");          /* PULL GPR9 FROM STACK   */
   asm (" addi r1,r1,12");         /* RESTORE STACK POINTER  */
}

getimmr()
{
   asm(" mfspr 3,638");
}

getevt()                           /* GET EVT LOCATION      */
{
   if ((getmsr() & 0x40) == 0)     /* IF MSR.IP IS 0        */
      return (0);                  /* THEN EVT IS IN LOW MEM*/
   else                            /* ELSE                  */
      return (0xFFF00000);         /* EVT IS IN HIGH MEM    */
}
```

ex1.c (2 of 2)

```
getmsr()                        /* GET MACHINE STATE REG VALUE  */
{
   asm(" mfmsr 3");             /* LOAD MACHINE STATE REG TO r3 */
}
```

MOTOROLA
Motorola Technical Training - **MPC860 Course**
Phoenix, Arizona

Title: **ex2.c**
Handling a Alignment Error Exception

Creation Date: **Jan. 10, 1996** From: **68360 Course**

Author: **Bob Bratt**

Description:

The results of this routine are:
1. Initializes the exception vector area with a service routine to increment an LED counter each time an alignment error occurs.
2. The exception service routine is made recoverable.
Assumptions:
1. Reset conditions exist.

Objective:

If the program executes properly, the LED counter contains a random count.

Equipment:

MPC860ADS board and a UDLP1.

UDLP1 Switch Settings: **N/A**

Connections: **MPC860ADS board and a UDLP1 are connected through P13.**

Updates:

ex2.c (1 of 2)

```
/* (EX2.C)                                                    */
#include "mpc860.h"                 /* DUAL PORT RAM EQUATES*/
struct dprbase *pdpr;               /* PNTR TO DUAL PORT RAM*/

main()
{
   void esr();                      /* EXCEPTION SERVICE RTN */
   int *ptrs,*ptrd;                 /* SOURCE & DEST POINTERS*/

   pdpr = (struct dprbase *) (getimmr() & 0xFFFF0000);
                                    /* INIT PNTR TO DPRBASE  */
   ptrs = (int *) esr;              /* INIT SOURCE POINTER   */
   ptrd = (int *)(getevt() + 0x600); /* INIT DEST POINTER    */
   do                               /* MOVE ESR TO EVT       */
      *ptrd++ = *ptrs;              /* MOVE UNTIL            */
   while (*ptrs++ != 0x4c000064);   /* RFI INTRUCTION        */
   pdpr->PDDAT = 0;                 /* CLEAR PORT D DATA REG */
   pdpr->PDDIR = 0xff;              /* MAKE PORT D8-15 OUTPUT*/
   asm(" li r21,0x1001");           /* INIT r21 TO UNALIGNED */
   asm(" lwarx r20,r0,r21");        /* ALIGNMENT INTERRUPT   */
}

#pragma interrupt esr
void esr()
{
   asm (" stwu r9,-12(r1)");       /* PUSH GPR9 ONTO STACK   */
   asm (" mfspr r9,26");           /* PUSH SRR0 ONTO STACK   */
   asm (" stw r9,4(r1)");
   asm (" mfspr r9,27");           /* PUSH SRR1 ONTO STACK   */
   asm (" stw r9,8(r1)");
   asm (" mtspr 80,0");            /* ENABLE INTERRUPTS      */
   pdpr->PDDAT += 1;
   asm (" mtspr 82,0");            /* MAKE NON-RECOVERABLE   */
   asm (" lwz r9,8(r1)");          /* PULL SRR1 FROM STACK   */
   asm (" mtspr 27,r9");
   asm (" lwz r9,4(r1)");          /* PULL SRR0 FROM STACK   */
   asm (" mtspr 26,r9");
   asm (" lwz r9,0(r1)");          /* PULL GPR9 FROM STACK   */
   asm (" addi r1,r1,12");         /* RESTORE STACK POINTER  */
}

getimmr()
{
   asm(" mfspr 3,638");
}

getevt()                           /* GET EVT LOCATION      */
{
   if ((getmsr() & 0x40) == 0)     /* IF MSR.IP IS 0        */
      return (0);                  /* THEN EVT IS IN LOW MEM*/
   else                            /* ELSE                  */
      return (0xFFF00000);         /* EVT IS IN HIGH MEM    */
}
```

ex2.c (2 of 2)

```
getmsr()                        /* GET MACHINE STATE REG VALUE  */
{
   asm(" mfmsr 3");             /* LOAD MACHINE STATE REG TO r3 */
}
```