MPC8xx 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

<table>
<thead>
<tr>
<th>CLASS</th>
<th>EXCEPTION TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYNCHRONOUS, UNORDERED</td>
<td>RESET, NON-MASKABLE</td>
</tr>
<tr>
<td>SYNCHRONOUS, UNORDERED</td>
<td>MACHINE CHECK (BUS ERROR)</td>
</tr>
<tr>
<td>ASYNCHRONOUS, ORDERED</td>
<td>EXTERNAL INTERRUPT DECREMENTER, PIT INTERRUPTS</td>
</tr>
<tr>
<td>SYNCHRONOUS (ORDERED, PRECISE)</td>
<td>INSTRUCTION -CAUSED EXCEPTIONS</td>
</tr>
</tbody>
</table>

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)**

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Reset Interrupt</strong></td>
<td>Occurs when the NMI pin is asserted, SWT times out, Hard or Soft Reset pins are asserted</td>
</tr>
<tr>
<td><strong>Machine Check Interrupt</strong></td>
<td>The accessed address does not exist or a data error was detected</td>
</tr>
<tr>
<td><strong>Data Storage Interrupt</strong></td>
<td>Never generated by the hardware</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Instruction Storage Interrupt</strong></td>
<td>Never generated by the hardware</td>
</tr>
<tr>
<td></td>
<td>The software may branch to this location as a result of an Implementation Specific Instruction TLB error interrupt</td>
</tr>
<tr>
<td><strong>Alignment Interrupt</strong></td>
<td><strong>Occurs as a result of one of the following cases:</strong></td>
</tr>
<tr>
<td></td>
<td>The operand of a Floating-Point load or store is not word aligned.</td>
</tr>
<tr>
<td></td>
<td>The operand of Load/Store multiple is not word aligned.</td>
</tr>
<tr>
<td></td>
<td>The operand of lwax or stwcx. is not word aligned.</td>
</tr>
<tr>
<td></td>
<td>The operand of Load/Store individual scalar instruction is not naturally aligned when MSR_LE = 1.</td>
</tr>
<tr>
<td></td>
<td>An attempt to execute multiple/string instruction is made when MSR_LE = 1</td>
</tr>
<tr>
<td><strong>Program Interrupt</strong></td>
<td><strong>Floating-Point Enabled Exception</strong> type Program interrupt is not generated by the MPC8xx core.</td>
</tr>
<tr>
<td></td>
<td><strong>Illegal Instruction</strong> type program interrupt is not generated by the Core, an Implementation Dependent Software Emulation Interrupt is generated instead</td>
</tr>
<tr>
<td></td>
<td><strong>Privileged instruction</strong> 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_p=1 and MSR_pr=1, as well as an attempt to execute privileged instruction when MSR_pr=1.</td>
</tr>
<tr>
<td><strong>Floating Point Unavailable Interrupt</strong></td>
<td>Not generated by the MPC8xx core.</td>
</tr>
<tr>
<td></td>
<td>An Implementation Dependent Software Emulation Interrupt will be taken on any attempt to execute Floating-Point instruction regardless of MSR_FP</td>
</tr>
</tbody>
</table>
### Exception Definitions (2 of 2)

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trace Interrupt</strong></td>
<td>Occurs if MSR\textsubscript{SE} = 1 and any instruction except rfi is successfully completed, or MSR\textsubscript{BE} = 1 and a branch is completed.</td>
</tr>
<tr>
<td><strong>Floating Point Assist Interrupt</strong></td>
<td>Not generated by the MPC8xx. An Implementation Dependent Software Emulation Interrupt will be taken on any attempt to execute Floating-Point instruction.</td>
</tr>
<tr>
<td><strong>Implementation Dependent Software Emulation Interrupt</strong></td>
<td>Occurs as a result of one of the following cases:</td>
</tr>
<tr>
<td></td>
<td>• When there is an attempt to execute any non implemented instruction. (This include all Illegal and unimplemented optional instructions and all floating point instructions).</td>
</tr>
<tr>
<td></td>
<td>• When there is an attempt to execute a mtspr or mfspr which specifies on core non implemented register. (regardless of spr\textsubscript{0}).</td>
</tr>
<tr>
<td></td>
<td>• When there is an attempt to execute a mtspr or mfspr which specifies off core non implemented register and spr\textsubscript{0} = 0 or MSR\textsubscript{PR} = 0.</td>
</tr>
<tr>
<td><strong>Implementation Specific Instruction TLB Miss Interrupt</strong></td>
<td>Occurs when MSR\textsubscript{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.</td>
</tr>
<tr>
<td><strong>Implementation Specific Instruction TLB Error Interrupt</strong></td>
<td>Occurs in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• The effective address cannot be translated (either Segment valid bit or Page valid bit of this page are cleared in the translation table).</td>
</tr>
<tr>
<td></td>
<td>• The fetch access violates storage protection.</td>
</tr>
<tr>
<td></td>
<td>• The fetch access is to Guarded storage and MSR\textsubscript{IR} = 1.</td>
</tr>
<tr>
<td><strong>Implementation Specific Data TLB Miss Interrupt</strong></td>
<td>Occurs when MSR\textsubscript{DR} = 1 and there is an attempt to access a page that its Effective Page Number can not be translated by the Data TLB.</td>
</tr>
<tr>
<td><strong>Implementation Specific Data TLB Error Interrupt</strong></td>
<td>Occurs in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• 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).</td>
</tr>
<tr>
<td></td>
<td>• The access violates the storage protection.</td>
</tr>
<tr>
<td></td>
<td>• An attempt to write to a page with negated Change Bit.</td>
</tr>
</tbody>
</table>
## Interrupt Priority Mapping

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

### Instruction Related Interrupt Detection Order

<table>
<thead>
<tr>
<th>#</th>
<th>Interrupt Type</th>
<th>Caused by</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Trace</td>
<td>Trace bit asserted</td>
</tr>
<tr>
<td>#2</td>
<td>Implementation Dependent Instruction TLB miss</td>
<td>Instruction MMU TLB Miss</td>
</tr>
<tr>
<td>#3</td>
<td>Implementation Dependent Instruction TLB error</td>
<td>Instruction MMU protection/translation error</td>
</tr>
<tr>
<td>#4</td>
<td>Machine Check Interrupt</td>
<td>Fetch Error</td>
</tr>
<tr>
<td>#5</td>
<td>Debug I- Breakpoint</td>
<td>Match detection</td>
</tr>
<tr>
<td>#6</td>
<td>Implementation Dependent Software Emulation Interrupt</td>
<td>Attempt to invoke un-implemented feature</td>
</tr>
<tr>
<td>#7</td>
<td>Floating-Point Unavailable</td>
<td>Attempt is made to execute Floating-Point instruction and MSRFP=0</td>
</tr>
<tr>
<td>#8</td>
<td>Privileged Instruction</td>
<td>Attempt to execute privileged instruction in problem mode</td>
</tr>
<tr>
<td>#9</td>
<td>Alignment Interrupt</td>
<td>Load store checking</td>
</tr>
<tr>
<td>#10</td>
<td>System Call Interrupt</td>
<td>SC Instruction</td>
</tr>
<tr>
<td>#11</td>
<td>Trap</td>
<td>Trap Instruction</td>
</tr>
<tr>
<td>#12</td>
<td>Debug -L Breakpoint</td>
<td>Match detection</td>
</tr>
<tr>
<td>#13</td>
<td>Implementation Dependent Data TLB miss</td>
<td>Data MMU TLB Miss</td>
</tr>
<tr>
<td>#14</td>
<td>Implementation Dependent Data TLB error</td>
<td>Dat MMU TLB Protection/translation error</td>
</tr>
<tr>
<td>#15</td>
<td>Machine Check Interrupt</td>
<td>Load or store access error</td>
</tr>
<tr>
<td>#16</td>
<td>Debug -L Breakpoint</td>
<td>Match detection</td>
</tr>
</tbody>
</table>
### Vector Table

**MSR - MACHINE STATE REGISTER**

<table>
<thead>
<tr>
<th>Vector Offset (HEX)</th>
<th>Exception Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>RESERVED</td>
</tr>
<tr>
<td>00100</td>
<td>SYSTEM RESET</td>
</tr>
<tr>
<td>00200</td>
<td>MACHINE CHECK</td>
</tr>
<tr>
<td>00300</td>
<td>DATA STORAGE</td>
</tr>
<tr>
<td>00400</td>
<td>INSTRUCTION STORAGE</td>
</tr>
<tr>
<td>00500</td>
<td>EXTERNAL INTERRUPT</td>
</tr>
<tr>
<td>00600</td>
<td>ALIGNMENT ERROR</td>
</tr>
<tr>
<td>00700</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>00800</td>
<td>FLOATING-POINT UNAVAILABLE</td>
</tr>
<tr>
<td>00900</td>
<td>DECREMENTER</td>
</tr>
<tr>
<td>00A00</td>
<td>RESERVED</td>
</tr>
<tr>
<td>00B00</td>
<td>RESERVED</td>
</tr>
<tr>
<td>00C00</td>
<td>SYSTEM CALL</td>
</tr>
<tr>
<td>00D00</td>
<td>TRACEx</td>
</tr>
<tr>
<td>00E00</td>
<td>FLOATING-POINT ASSISTx</td>
</tr>
<tr>
<td>01000</td>
<td>IMPLEMENTATION DEPENDENT SOFTWARE EMULATION</td>
</tr>
<tr>
<td>01100</td>
<td>IMPLEMENTATION DEPENDENT INSTRUCTION TLB MISS</td>
</tr>
<tr>
<td>01200</td>
<td>IMPLEMENTATION DEPENDENT DATA TLB MISS</td>
</tr>
<tr>
<td>01300</td>
<td>IMPLEMENTATION DEPENDENT INSTRUCTION TLB ERROR</td>
</tr>
<tr>
<td>01400</td>
<td>IMPLEMENTATION DEPENDENT DATA TLB ERROR</td>
</tr>
<tr>
<td>01500 - 01BFF</td>
<td>RESERVED</td>
</tr>
<tr>
<td>01C00</td>
<td>IMPLEMENTATION DEPENDENT DATA BREAKPOINT</td>
</tr>
<tr>
<td>01D00</td>
<td>IMPLEMENTATION DEPENDENT INSTRUCTION BREAKPOINT</td>
</tr>
<tr>
<td>01E00</td>
<td>IMPLEMENTATION DEPENDENT PERIPHERAL BREAKPOINT</td>
</tr>
<tr>
<td>01F00</td>
<td>IMPLEMENTATION DEPENDENT NON MASKABLE DEVELOPMENT PORT</td>
</tr>
</tbody>
</table>

**IP** = 0 Vector Table Address at 0x00000000

**IP** = 1 Vector Table Address at 0xFFFF0000.

- HARD & SRESETS
- **TEA** (BUS ERROR)
- **IRQ[1 :7], PIT, TB, RTC, PCMCIA, CPM**
- ALIGNMENT ERROR
- INTR. TRAPS, ERRORS, ILLEGAL, PRIVILEGED
- MSR[FPP]=0 & F.P. INSTRUCTION ENCOUNTERED
- DECREMENTER REGISTER
- ‘SC’ INSTRUCTION
- SINGLE-STEP OR BRANCH TRACING
- SOFTWARE ASSIST FOR INFREQUENT & COMPLEX FP OPERATIONS

For More Information On This Product, Go to: [www.freescale.com](http://www.freescale.com)
**MSR After Hard Reset**

**MSR - MACHINE STATE REGISTER**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RESVD</td>
</tr>
<tr>
<td>POW</td>
<td>ISF</td>
<td>ILE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE</td>
<td>PR</td>
<td>FP</td>
<td>ME</td>
<td>FE0</td>
<td>SE</td>
<td>BE</td>
<td>FE1</td>
<td>RESVD</td>
<td>IP</td>
<td>IR</td>
<td>DR</td>
<td>RESVD</td>
<td>RESVD</td>
<td>RI</td>
<td>LE</td>
</tr>
</tbody>
</table>

**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.
- **IP** * - **Interrupt Prefix . Vector Table Located**
  - at 0x000n - nnnn or at 0xFFFFn - 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 of the load or store for misaligned exceptions.

For More Information On This Product, Go to: www.freescale.com
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.

**SRR0 - Save and Restore Register 0**
- Holds address of next instruction to be executed

**SRR1 - Save and Restore Register 1**
- Holds copy of MSR before exception

**MSR - MACHINE STATE REGISTER**
- RESERVED
- Change to Exception Value

---

Freescale Semiconductor, Inc.

For More Information On This Product, Go to: www.freescale.com
How the RFI Instruction Operates

**EXCEPTION PROCESSING**

1. HW Saves Instr. Addr to SRR0
2. HW Saves MSR to SRR1
3. HW changes MSR (change to Supervisor Mode, mask other maskable exceptions...)

**NORMAL PROCESSING**

RFI Instruction is Supervisor-only used to restore previous Machine State.
How to Make the ESR Recoverable

Making the ESR Recoverable

```assembly
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

```assembly
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 Table

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>MSR&lt;sub&gt;EE&lt;/sub&gt;</th>
<th>MSR&lt;sub&gt;RI&lt;/sub&gt;</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIE (80)</td>
<td>1</td>
<td>1</td>
<td>External Interrupt Enable</td>
</tr>
<tr>
<td>EID (81)</td>
<td>0</td>
<td>1</td>
<td>External Interrupt Disable, but other interrupts are recoverable</td>
</tr>
<tr>
<td>NRI(82)</td>
<td>0</td>
<td>0</td>
<td>Non-Recoverable Interrupt</td>
</tr>
</tbody>
</table>
Title: ex1.c
Handling a System Call Exception

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

Author: Bob Brait

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:
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 (" mtsp 80,0"); /* ENABLE INTERRUPTS */
    pdpr->PDDAT += 1;
    asm (" mtsp 82,0"); /* MAKE NON-RECOVERABLE */
    asm (" lwz r9,8(r1)"); /* PULL SRR1 FROM STACK */
    asm (" mtsp 27,r9");
    asm (" lwz r9,4(r1)"); /* PULL SRR0 FROM STACK */
    asm (" mtsp 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 */
}
getmsr() /* GET MACHINE STATE REG VALUE */
{
    asm(" mfmsr 3"); /* LOAD MACHINE STATE REG TO r3 */
}
Title: 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:
#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)" );          /* PUSH SRR1 ONTO STACK */
        asm (" mfspr r9,27");           /* PUSH SRR0 ONTO STACK */
        asm (" stw r9,8(r1)" );          /* PUSH SRR0 ONTO STACK */
        asm (" mtsp 80,0");             /* ENABLE INTERRUPTS */
        pdpr->PDDAT += 1;
        asm (" mtsp 82,0");             /* MAKE NON-RECOVERABLE */
        asm (" lwz r9,8(r1)" );          /* PULL SRR1 FROM STACK */
        asm (" mtsp 27,r9" );            /* PULL SRR0 FROM STACK */
        asm (" lwz r9,4(r1)" );          /* PULL SRR0 FROM STACK */
        asm (" mtsp 26,r9" );            /* PULL SRR0 FROM STACK */
        asm (" lwz r9,0(r1)" );          /* PULL GPR9 FROM STACK */
        asm (" addi r1,r1,12" );         /* RESTORE STACK POINTER */
    }

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

    getevt()
    {
        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 */
    }
C

default_1(x)
{
    return x;
}

C

def Timeout
{
    int

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