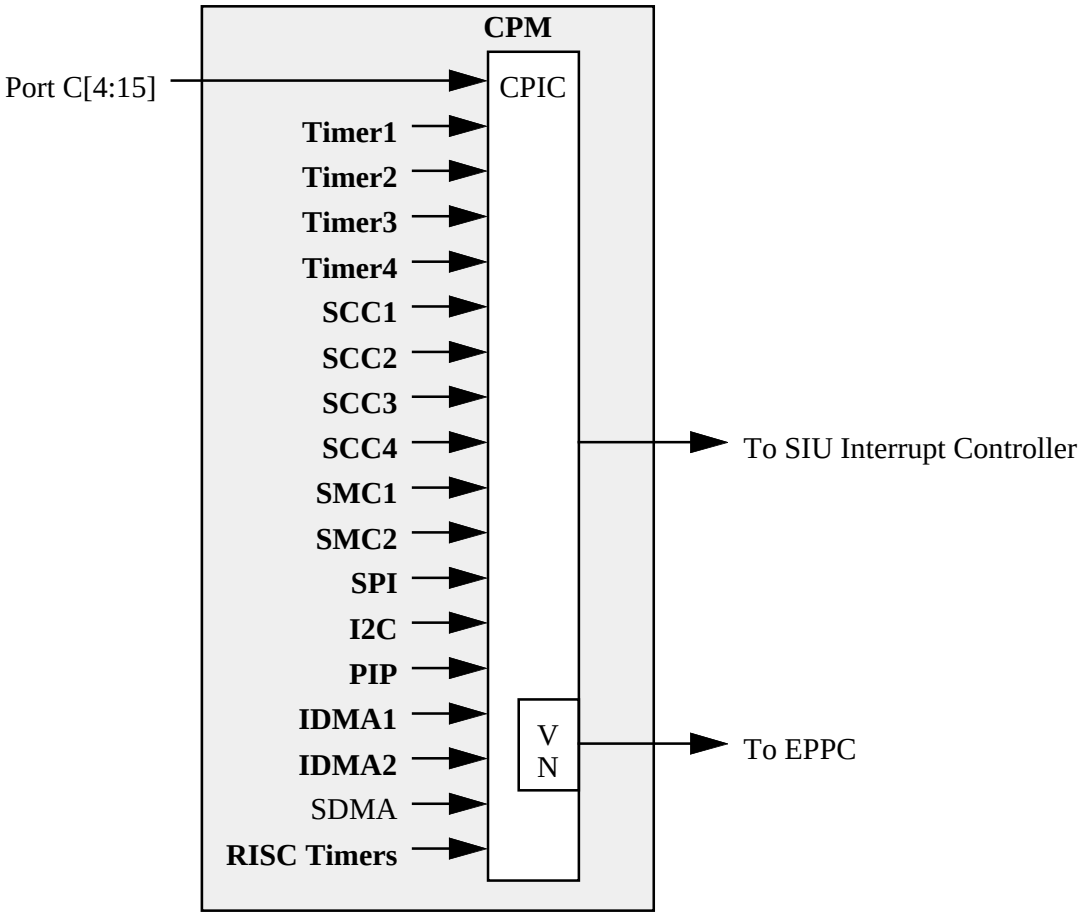
CPM Interrupt Controller

What is the CPIC?

Definition

The CPIC is the focal point for all interrupts associated with the CPM. It accepts and prioritizes all the internal and external interrupts from all functional blocks associated with the CPM.

Example



Bolded names are sub-block maskable interrupt sources.

CPIC Features

Important functions of the CPIC are:

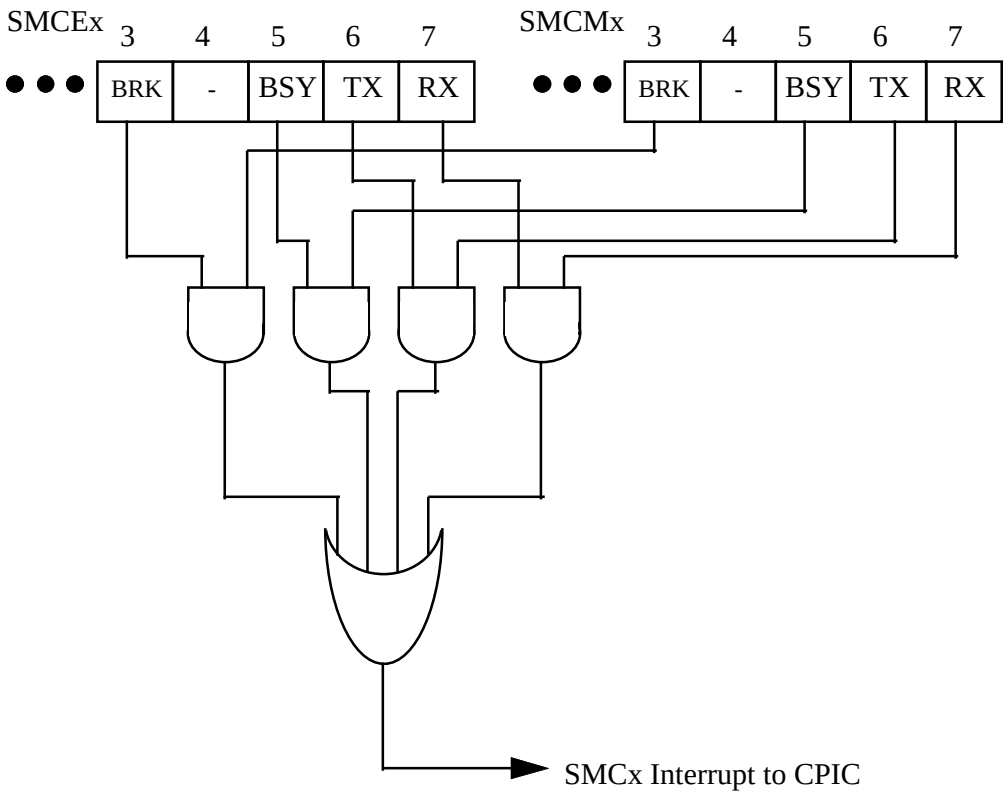
- Asserts an interrupt to the SIU interrupt controller at a user programmable level.
- Generates a unique vector number for each interrupt source.
- Prioritizes the interrupts for which it is responsible.
  - Highest priority interrupt source is programmable by the user.
  - Programmable priority between SCCs.
  - Two priority schemes for the SCCs.

What is a Sub-Block Maskable Interrupt?

Definition

If an interrupt source is maskable within the particular sub-block of which it is a part, it is referred to as sub-block maskable.

Example, SMCx



Programming Model

CICR - CPM Interrupt Configuration Register P. 814

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
								SCdP		SCcP		SCbP		SCaP	

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
IRL0_IRL2			HP0_HP4					IEN	-						SPS

CIPR - CPM Interrupt Pending Register P. 816

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PC15	SCC1	SCC2	SCC3	SCC4	PC14	Timer 1	PC13	PC12	SDMA	IDMA 1	IDMA 2	-	Timer 2	R_TT	I2C

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PC11	PC10	-	Timer 3	PC9	PC8	PC7	-	Timer 4	PC6	SPI	SMC1	SMC2 /PIP	PC5	PC4	-

CIMR - CPM Interrupt Mask Register P. 816

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PC15	SCC1	SCC2	SCC3	SCC4	PC14	Timer 1	PC13	PC12	SDMA	IDMA 1	IDMA 2	-	Timer 2	R_TT	I2C

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PC11	PC10	-	Timer 3	PC9	PC8	PC7	-	Timer 4	PC6	SPI	SMC1	SMC2 /PIP	PC5	PC4	-

CISR - CPM In-Service Register P. 816

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PC15	SCC1	SCC2	SCC3	SCC4	PC14	Timer 1	PC13	PC12	SDMA	IDMA 1	IDMA 2	-	Timer 2	R_TT	I2C

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PC11	PC10	-	Timer 3	PC9	PC8	PC7	-	Timer 4	PC6	SPI	SMC1	SMC2 /PIP	PC5	PC4	-

CIVR - CPM Interrupt Vector Register P. 814

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
VN					0										IACK

How to Prioritize the SCCs (1 of 2)

Introduction

The SCCs must be prioritized relative to each other. The user controls the order of priority in the CICR, fields SCdP, SCcP, SCbP, and SCaP.

Priority Matrix

		Lowest			Highest	**Priority**
SCC	**Code**	SCdP	SCcP	SCbP	SCaP	**CICR**
SCC1	00					
SCC2	01					
SCC3	10					
SCC4	11					

Example

Problem: Set the priority so that SCC1 is highest, SCC3 is second hightest, SCC2 second lowest, and SCC4 the lowest.

		Lowest			Highest	**Priority**
SCC	**Code**	SCdP	SCcP	SCbP	SCaP	**CICR**
SCC1	00				00	
SCC2	01		01			
SCC3	10			10		
SCC4	11	11				

```
pdpr->CICR.SCaP = 0;
pdpr->CICR.SCbP = 2;
pdpr->CICR.SCcP = 1;
pdpr->CICR.SCdP = 3;
```

Exercise

Set the priority so that SCC2 is the highest priority, SCC3 is second highest, SCC4 is the second lowest, and SCC1 is the lowest.

```
pdpr->CICR.SCaP = _;
pdpr->CICR.SCbP = _;
pdpr->CICR.SCcP = _;
pdpr->CICR.SCdP = _;
```

Comment

SCaP, SCbP, SCcP, and SCdP should all have different numbers.

How to Prioritize the SCCs (2 of 2)

Introduction

In addition to being prioritized relative to each other, the SCCs can be grouped together in the priority list or spread out.

Grouped Priority

Priority	Interrupt Source
Highest	PC15
	SCCa
	SCCb
	SCCc
	SCCd

pdpr->CICR.SPS = 0;

Spread Priority

Priority	Interrupt Source
Highest	PC15
	SCCa
	SCCb
	SCCc
	SCCd

pdpr->CICR.SPS = 1;

How to Specify the Highest Priority Interrupt Source

Introduction

The user must specify which interrupt source is to be given top priority. This is done by writing the 5-bit interrupt vector number to CICR.HP0_HP4. A priority list is on p. 810.

Example

Problem: make the SDMA interrupt the highest priority.

```
pdpr->CICR.HP0_HP4 = 0x16;
```

Exercise

Make PC15 the highest priority interrupt..

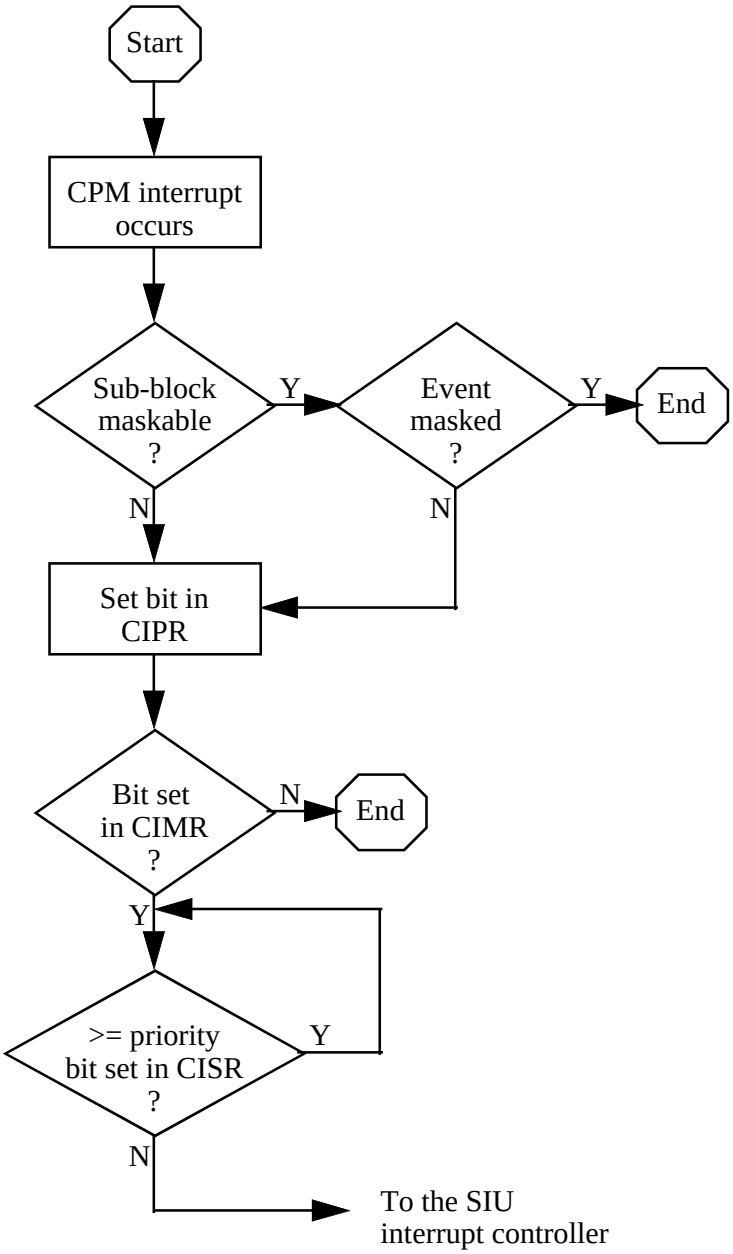
```
pdpr->CICR.HP0_HP4 = ____;
```

How the CPIC Processes an Interrupt Input

Introduction

The CPIC receives an interrupt from one of its 29 sources, processes it, and, assuming no masking, asserts its programmed interrupt level to the SIU interrupt controller.

Flow Diagram of How the CPIC Processes an Interrupt

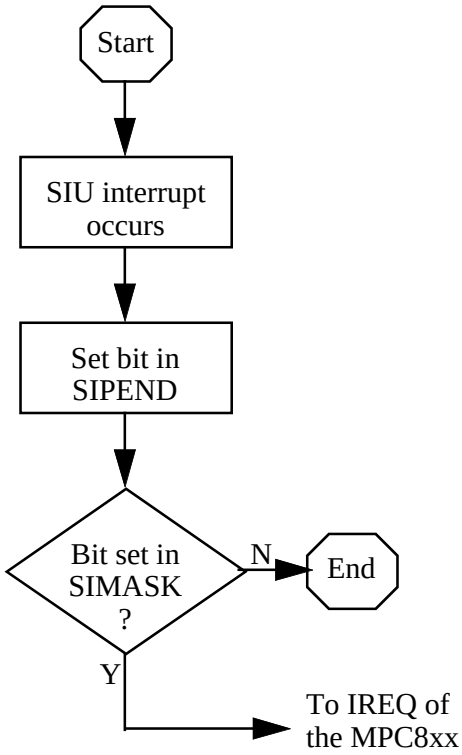


How the SIU Processes an Interrupt Input

Introduction

The SIU receives an interrupt from one of 8 external sources or 1 of 8 internal sources and, assuming no masking, assertes the IREQ input to the MPC8xx.

Flow Diagram of How the SIU Processes an Interrupt



MPC8xx Action

Following the assertion of IREQ, the MPC8xx completes the present instruction and program control goes to offset 0x500 in the exception vector table.

How to Initialize a CPM for Interrupts

Introduction

Here we describe the steps in initializing the CPM on the MPC860 for interrupts.

Assumptions

- IMMR has been initialized previously. If not, the user must initialize it.
- Except for the above, reset conditions exist.

Action

Here are the steps in initialization :

Step	Action	Example
1	Initialize CPM Intrpt Config Reg, CICR SCdP: lowest priority SCC SCcP: 2nd lowest priority SCC SCbP: 2nd highest priority SCC SCaP: highest priority SCC IRL0_IRL2: CPM intrpt level HP0_HP4: highest priority intrpt source SPS: spread priority (814)	pdpr->CICR.HP0_HP4 = 0x16; /* SDMA HIGHEST PRIORITY INTERRUPT */
2	Initialize Interrupt Mask Reg, CIMR SCC1-4 PC4-15 TIMER1-4 IDMA1-2 SMC1-2 SDMA R-TT SPI I2C (817)	pdpr->CIMR.SCC2 = 1; /* ENABLE SCC2 INTRPTS */
3	Initialize SI Edge/Level Reg, SIEL EDx:edge or level interrupt input WMx:exit low power mode where x is 0 to 7 (191)	pdpr->SIEL.WM5 = 1; /*WAKEUP 860 FOR LEVEL 5 INTERRUPT*/
4	Initialize SI Mask Reg, SIMASK IRMx:enable external interrupt input LVMx:enable internal interrupt input where x is 0 to 7 (191)	pdpr->SIMASK.ASTRUCT.IRM6 = 1; /*ENABLE IRQ6 INTERRUPTS */
5	Enable CPM Interrupts (814)	pdpr->CICR.IEN = 1; /* ENABLE CPM INTERRUPTS */
6	Initialize Enable Interrupts, EIE	asm (“ mtspr 80,0”);; /* ENABLE INTERRUPTS */

How to Handle a CPIC Interrupt (1 of 2)

First Steps in Servicing CPM Interrupts

• The first steps in servicing a CPM interrupt:

1	Read the interrupt code in the SI vector register, SIVEC, and go to service routine for that code. (192)	if (pdpr->SIVEC.IC == 0x38)
		  irq7esr();
		/* IF IRQ7, GO TO IRQ7ESR */

2	Clear the service bit in the SI Pending Register, SIPEND (191)	pdpr->SIPEND = 1<<(31-6); /* CLEAR IRQ3 PENDING BIT*/
3	Required only if service routine is to be recoverable and lower priority interrupts are to be masked. Save the SI mask reg, SIMASK Mask lower interrupt levels (191)	sptr++ = pdpr->SIMASK.ASINT; /* STACK SIMASK REG */ pdpr->SIMASK.ASINT &= 0xF0000000; /* MASK INTRPTS 2-7 */
4	Required only if service routine is to be recoverable. Save SRR0 & SRR1 on the stack Enable interrupts	asm (“ mfspr r9,26”); asm (“ stwu r9,-8(r1)”); asm (“ mfspr r9,27”); asm (“ stw r9,4(r1)”); asm (“ mtspr 80,0”);
5	Request the vector number via the CPM Interrupt Vector Reg, CIVR (819)	pdpr->CIVR.IACK = 1; /* REQUEST VECTOR NUMBER*/

6	Read the interrupt vector in the CPM interrupt vector reg, CIVR, and go to service routine for that vector number. (819)	if (pdpr->CIVR.VN == 0x10
		  i2cesr();
		/* I2C VEC NUM, GO TO I2CESR*/

7	If this is a submodule maskable event source, read the event register.	er = pdpr->SCCE2; /* GET EVENT REGISTER */
8	If this is a submodule maskable event source, clear the known events.	pdpr->SCCE2 = er; /* CLEAR EVENT REGISTER */

How to Handle a CPIC Interrupt (2 of 2)

Last Steps in Servicing CPM Interrupts

• The last steps in servicing a CPM interrupt:

1	Clear the bit in the in-service reg, CISR (818)	pdpr->CISR = 1<<(31-6); /* CLEAR TIMER 1 BIT */
2	Required only if service routine was made recoverable. Disable interrupts Restore SRR0 & SRR1 on the stack	asm (“ mtspr 82,0”); asm (“ lwz r9,4(r1)”); asm (“ mtspr 27,r9”); asm (“ lwz r9,0(r1)”); asm (“ addi r1,r1,8;”) asm (“ mtspr 26,r9”);
3	Required only if service routine was made recoverable and lower priority interrupts were masked. Restore the SI mask reg, SIMASK	pdpr->SIMASK.ASINT = --sptr; /* RESTORE SIMASK REG */

FREESCALE
Freescale Technical Training - **MPC860 Course**
Phoenix, Arizona

Title: **pc8.c**
Handling an 860 CPM Interrupt

Creation Date: **Jan. 10, 1996** From: **MC68360 Course**

Author: **Bob Bratt**

Description:

The results of this routine are:
1. Initializes the exception vector area with a service routine.
2. The service routine jumps to a function based on the interrupt code.
3. The function increments a counter each time an external interrupt level 1 occurs.
Assumptions:
1. IMMR has been previously initialized.
2. Except for 1, reset conditions exist.

Objective:

If the program executes properly, the LED counter is equal to the number of times that the black button on the UDLP1 has been pressed.

Equipment:

MPC860ADS board and UDLP1

UDLP1 Switch Settings: **N/A**

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

Updates:

pc8.c (1 of 2)

```
/* Equipment : 860ADS Evaluation Board and                    */
/*             UDLP1 Universal Development Lab Board          */
/*             Pins 2 and 3 of JP2 must be jumpered           */
/* Connected: P10-C15 of ADS to J4-11 of UDLP1                */
/* (PC8.C)                                                    */

#include "mpc860.h"                 /* DUAL PORT RAM EQUATES */
struct dprbase *pdpr;               /* PNTR TO DUAL PORT RAM */
static int buffer[10];              /* STACK BUFER FOR SIMASK*/
static int sp = 0;                  /* STACK BUFFER POINTER  */

main()
{
   void intbrn();                   /* EXCEPTION SERVICE RTN */
   int *ptrs,*ptrd;                 /* SOURCE & DEST POINTERS*/
   char intlvl = 4;                 /* INTERRUPT LEVEL       */

   pdpr = (struct dprbase *) (getimmr() & 0xFFFF0000);
                                    /* INIT PNTR TO DPRBASE  */
   ptrs = (int *) intbrn;           /* INIT SOURCE POINTER   */
   ptrd = (int *)(getevt() + 0x500); /* INIT DEST POINTER    */
   do                               /* MOVE ESR TO EVT       */
      *ptrd++ = *ptrs;              /* MOVE UNTIL            */
   while (*ptrs++ != 0x4c000064);   /* RFI INTRUCTION        */
   pdpr->CICR.IRL0_IRL2 = (unsigned) (intlvl);
                                    /* CPM INTERRUPTS LEVEL 4*/
   pdpr->CICR.HP0_HP4 = 0x1F;       /* NO INT PRIORITY CHANGE*/
   pdpr->PDDAT = 0;                 /* CLEAR PORT D DATA REG */
   pdpr->PDDIR = 0xff;              /* MAKE PORT D8-15 OUTPUT*/
   pdpr->PCINT |= 0x80;             /* CONFIG PC8 INTRPT EDGE*/
   pdpr->CIMR.PC8 = 1;              /* ENABLE PORT C,8 INTRPT*/
   pdpr->SIMASK.ASTRUCT.LVM4 = 1;   /* ENABLE IRQ4 INTERRUPTS*/
   pdpr->CICR.IEN = 1;              /* ENABLE CPM INTERRUPTS */
   asm(" mtspr 80,0");              /* ENABLE INTERRUPTS     */
   while (1==1);
}

#pragma interrupt intbrn
void intbrn()
{
   void cpmesr();

   asm (" stwu r9,-4(r1)");        /* PUSH GPR9 ONTO STACK  */
   switch (pdpr->SIVEC.IC)         /* PROCESS INTERRUPT CODE*/
   {
      case 0x24: asm (" mfspr r9,8"); /* PUSH LR ONTO STACK */
              asm (" stwu r9,-4(r1)");
              asm (" bla cpmesr");  /* PROCESS IRQ1 CODE    */
              asm (" lwz r9,0(r1)"); /* PULL LR FROM STACK  */
              asm (" addi r1,r1,4"); /* RESTORE STACK POINTR*/
              asm (" mtspr 8,r9");
      break;
      default:;
   }
```

pc8.c (2 of 2)

```
    asm (" lwz r9,0(r1)");          /* PULL GPR9 FROM STACK  */
    asm (" addi r1,r1,4");          /* RESTORE STACK POINTER */
}

void cpmesr()
{
    unsigned v1;                     /* TEMPORARY STORAGE     */
    pdpr->CIVR.IACK = 1;             /* REQUEST VECTOR NUMBER */
    v1 = pdpr->CIVR.VN;              /* COPY VECTOR NUMBER    */
    buffer[sp++] = pdpr->SIMASK.ASINT;       /* STACK SIMASK  */
    pdpr->SIMASK.ASINT &= 0xFFC<<(31-9);      /* MASK INTS 5-7*/
    asm (" mfspr r9,26");             /* PUSH SRR0 ONTO STACK */
    asm (" stwu r9,-8(r1)");
    asm (" mfspr r9,27");             /* PUSH SRR1 ONTO STACK */
    asm (" stw r9,4(r1)");
    asm (" mtspr 80,0");            /* ENABLE INTERRUPTS      */
    switch (v1)                      /* PROCESS VECTOR NUMBER */
    {
       case 0xA:                     /* PC8 VECTOR NUMBER     */
          pdpr->PDDAT += 1;          /* INCREMENT DISPLAY     */
          pdpr->CISR = 1<<(31-21);       /* CLEAR IN-SRVCE BIT*/
          break;
       default:;
    }
    asm (" mtspr 82,0");            /* MAKE NON-RECOVERABLE   */
    asm (" lwz r9,4(r1)");          /* PULL SRR1 FROM STACK   */
    asm (" mtspr 27,r9");
    asm (" lwz r9,0(r1)");          /* PULL SRR0 FROM STACK   */
    asm (" addi r1,r1,8");
    asm (" mtspr 26,r9");
    pdpr->SIMASK.ASINT = buffer[--sp]; /* RESTORE SIU MASK REG  */
}

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 */
}
```