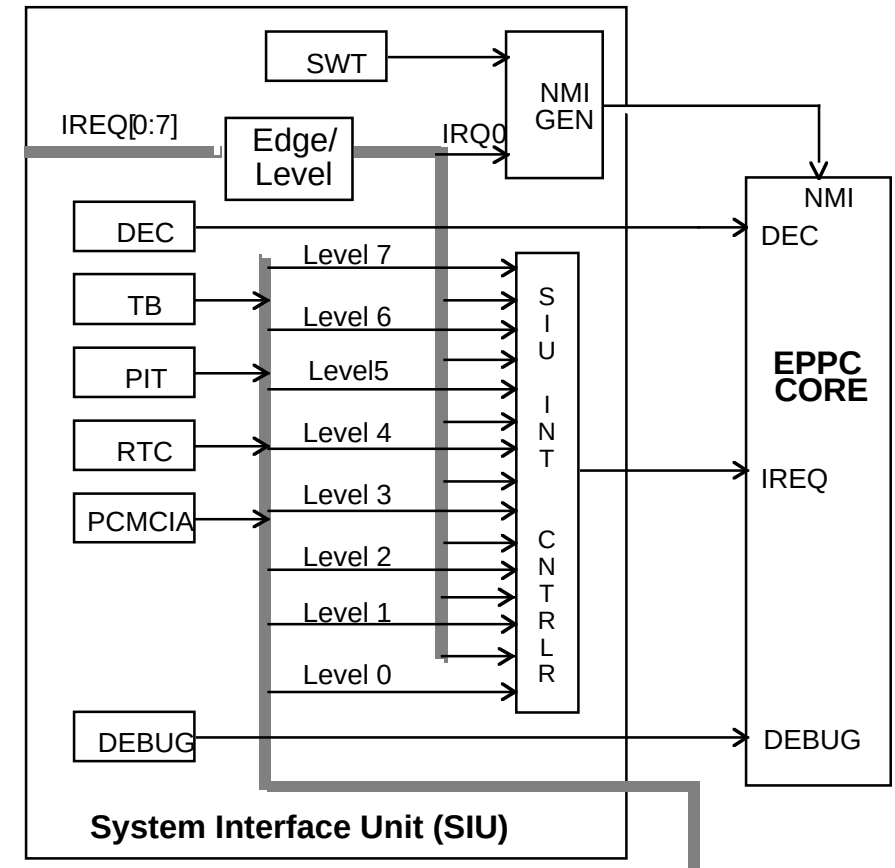
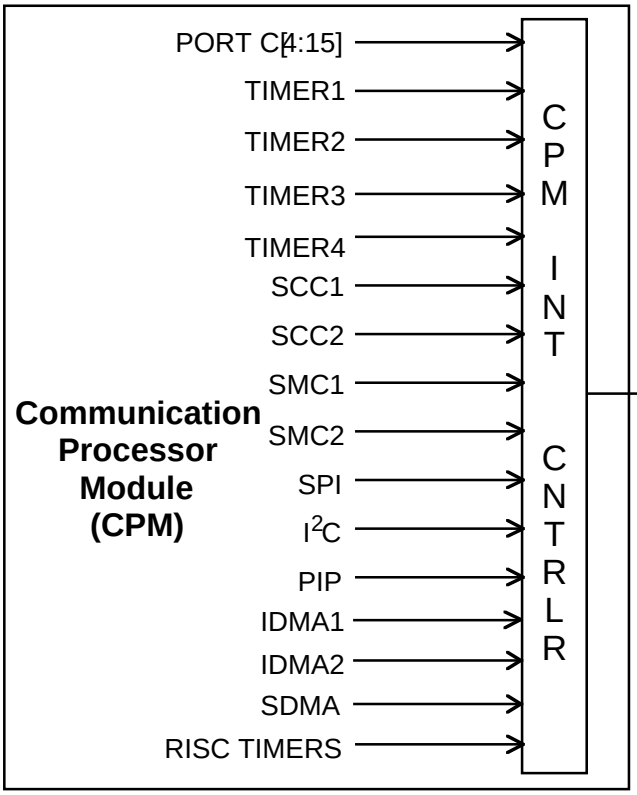
SIU Interrupt Controller

SIU Interrupts



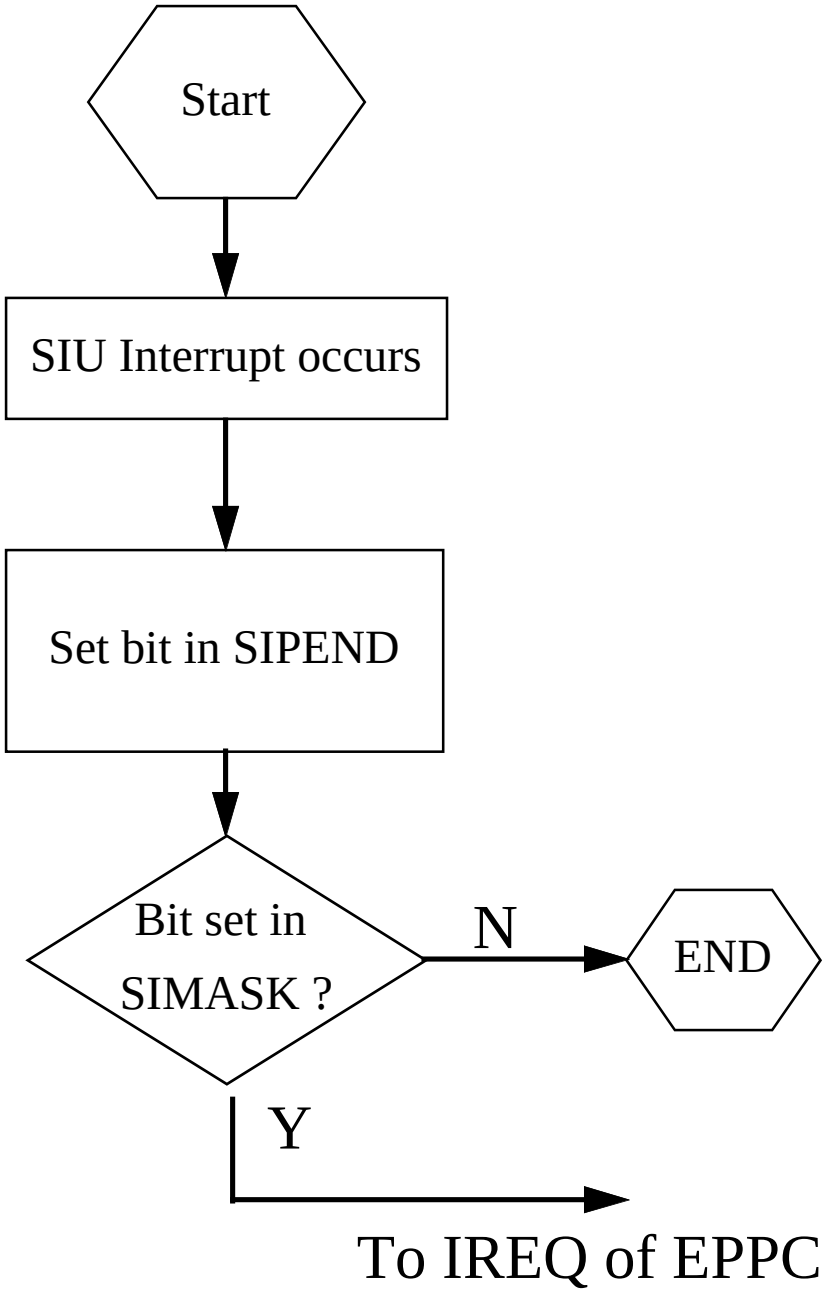
- SIU Receives an interrupt from 1 of 8 external sources or 1 of 8 internal sources
  - Assuming no masking
- SIU asserts the IREQ input to the EPPC Core



To SIU Interrupt Controller

- CPIC Generates an interrupt to the SIU Interrupt Controller at a User Programmable Level

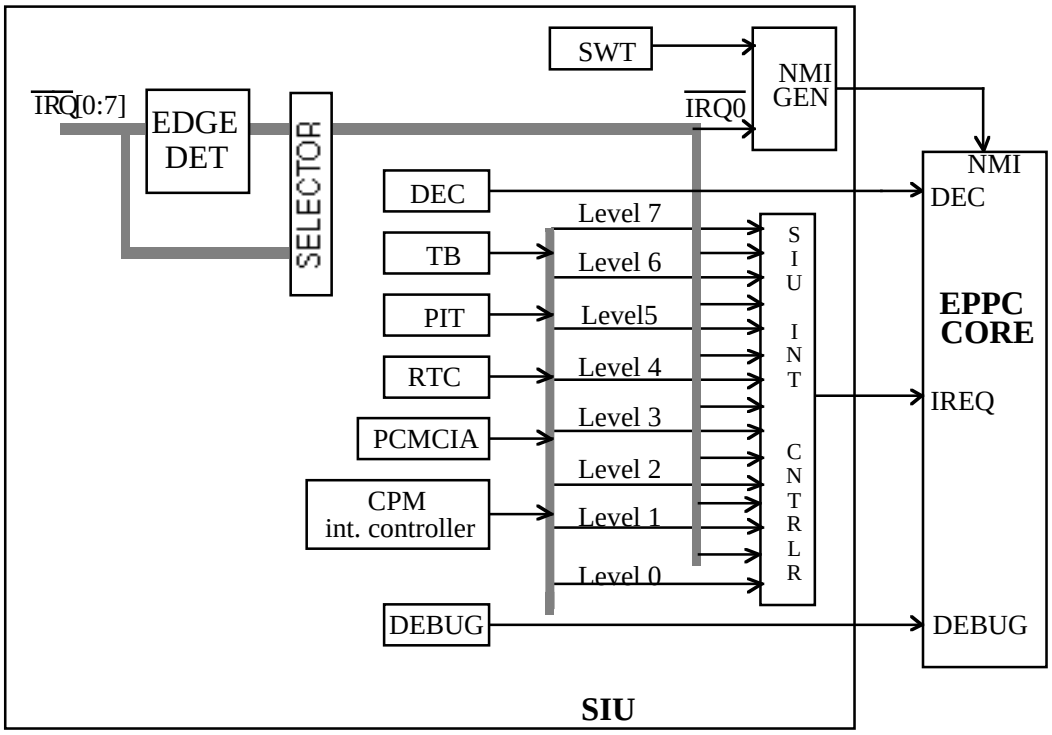
Flow Diagram: How SIU Processes an Interrupts



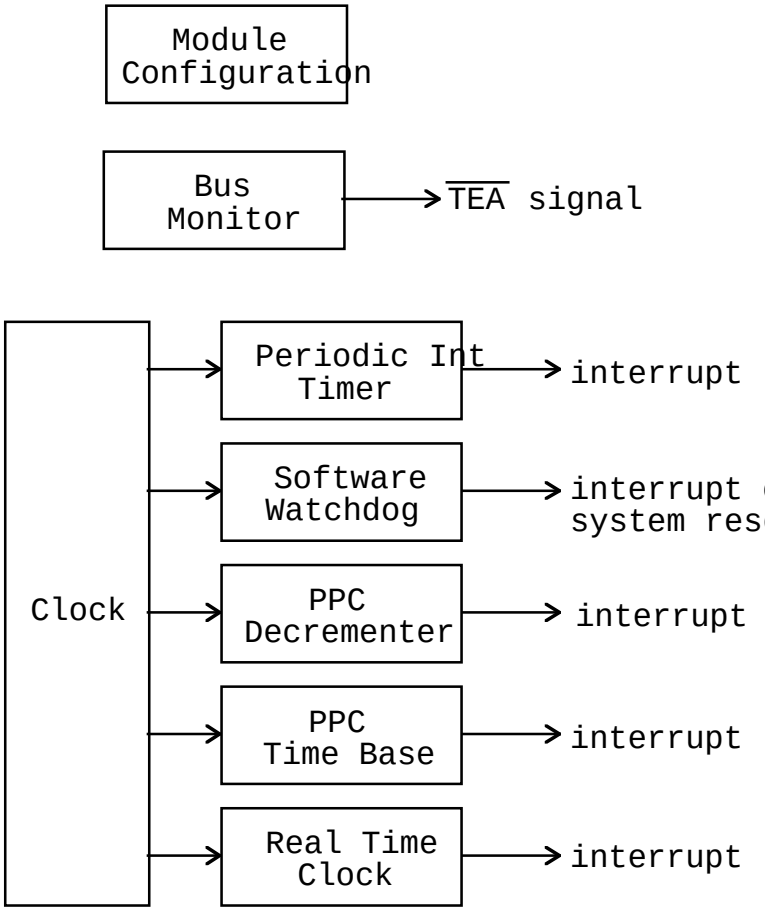
Vector Table

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
0 0800	FLOATING-POINT UNAVAILABLE	MSR[FP]=0 & F.P. INSTRUCTION ENCOUNTERED
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	

SIU Interrupt Sources



System Configuration and Protection Logic



Priority of SIU Interrupt Sources

Number	Priority Level	Interrupt Source Description	Interrupt Code
0	Highest	IRQ0	00000000
1		LEVEL 0	00000100
2		IRQ1	00001000
3		LEVEL 1	00001100
4		IRQ2	00010000
5		LEVEL 2	00010100
6		IRQ3	00011000
7		LEVEL 3	00011100
8		IRQ4	00100000
9		LEVEL 4	00100100
10		IRQ5	00101000
11		LEVEL 5	00101100
12		IRQ6	00110000
13		LEVEl 6	00110100
14		IRQ7	00111000
15	Lowest	LEVEL 7	00111100
16 -31		Reserved	

- **IRQx Pins have a fixed priority level**

  Example: IRQ0 Highest Priority Level (Code = 00000000)

  IRQ7 Lowest Priority Level (Code = 00111000)

- **Interrupt Code(SIVEC reg) defines interrupt priority level**
- **IRQx Interrupt Codes are only associated with IRQx pins**
- **IRQx interrupt source has highest priority than LEVELx**

  Example: If both IRQ3 and LEVEL3 interrupt simultaneously

  IRQ3 will be acknowledged first

Interrupt Sources vs. Levels

- Each interrupt source is identified by a level
- PIT,RTC, TB, CPM levels are assigned using register PISCR, RTCSC, TBSCR, CICR respectively(disscussed later)

Source	Level
NMI:	
SWT	
IRQ0_L	
External Pins:	
IRQ1_L	Level 1
IRQ2_L	Level 2
IRQ3_L	Level 3
IRQ4_L	Level 4
IRQ5_L	Level 5
IRQ6_L	Level 6
IRQ7_L	Level 7
Internal:	
PIT	Assignable between 0:7
RTC	Assignable between 0:7
TB	Assignable between 0:7
CPM	Assignable between 0:7

Identifying an Interrupt Level

- When an interrupt occurs, the bit corresponding to it's level is set in the SIPEND register.

SIPEND Register (Interrupt Pending)

- Other interrupt registers we will use include:

  - **SIMASK: 32 bit register (Read/Write)**
    - » enables or disables each of the 8 interrupt levels and 8 IRQ Lines
  - **SIEL: 32 bit register (Read/Write)**
    - » specifies whether a falling edge or a low logical level in the IRQ line will be detected as an interrupt request
    - » specifies if MPC860 to exit low power mode with a detection of an interrupt request
  - **SIVEC: 32 bit register (Read Only)**
    - » Contains an 8 bit Interrupt code representing the unmasked interrupt source which has the highest priority level

Interrupt Priorities

- All interrupts share the same exception vector ***(500)***
- Levels and IRQ lines can be used to efficiently determine priorities.
  - IRQ0 = highest priority
  - Level 7 = lowest priority

Interrupt Initialization Steps - 1

1. Assign interrupt levels to the PIT,RTC, TB, CPM
   - IRQ0:7 pins have fixed levels
   - PIT,RTC,TB, CPM levels are assigned using register PISCR, RTCSC,TBSCR, CICR respectively

PISCR Register (Periodic Interrupt Status/Control Register)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PIRQ - Periodic Interrupt Request Level								PS	RESERVED				PIE	PITF	PTE
Reset: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RTCSC Register (Real Time Clock Status/Control Register)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RTCIRQ - RTC Interrupt RequestLevel								SEC	ALR		38K	SIE	ALE	RTF	RTE
Reset: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TBSCR Register (Time Base Status/Control Register)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TBIRQ - Time Base Interrupt Request								REFA	REFB	RESERVED		REFAE	REFBE	TBF	TBE
Reset: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

–To specify a certain level, the appropiate one of these bits should be set.

CICR Register (CPM Interrupt Configuration Register - 24 bits)

8	9	10	11	12	13	14	15	16	17	18	19
SCdP		SCcP		SCbP		ScaP		IRL0	IRL1	IRL2	HP0
Reset:								0	0	0	

IRL0 - IRL2 : Interrupt Request Level

Level 0 --> Highest Priority

Level 7 --> Lowest Priority

** Value $4 is a good value to choose for most systems

Interrupt Initialization Steps - 2

2. Assign IRQ pins as interrupts and edge or level sensitive

SIEL Register (External Interrupt Level/Edge Sensitive Register)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ED0	WM0	ED1	WM1	ED2	WM2	ED3	WM3	ED4	WM4	ED5	WM5	ED6	WM6	ED7	WM7

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESERVED															

EDx Bit

1 = IRQx “falling” edge sensitive
0 = IRQx “low” logical level sensitive

WMx Bit (Wake up Mask Bit)

1 = MPC860 to exit low power mode with a detection of an interrupt request on corresponding line

3. Enable Interrupts to CPU core

SIMASK Register (Interrupt Mask Register)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IRM0	LVM0	IRM1	LVM1	IRM2	LVM2	IRM3	LVM3	IRM4	LVM4	IRM5	LVM5	IRM6	LVM6	IRM7	LVM7
Reset: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESERVED															
Reset: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IRMx - External Interrupt Request Mask
LVMx - Level Interrupt Request Mask

1 = Enables the generation of an interrupt request to the CPU core

SIMASK updated by software and cleared upon reset

Interrupt Table Handling Example

SIVEC Register (Read Only)

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

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Interrupt Code:** 8 bit code representing the umasked interrupt source which has the highest priority level.
  - –If read as a “BYTE” a “Branch table” can be used in which each entry contains 1 instruction (BRANCH)
  - –If read as a “HALF WORD” each entry may contain a full routine up to 256 instructions

Example:

```
INTR:•••

     Save state
     R3 <- @SIVEC
     R4 <-- Base of branch table

     •••

     lbz  RX, R3 (0)# load as byt
     add  RX, RX, R4
     mtsprCTR, RX
     bctr
```

BASE	b Routine1
BASE + 4	b Routine2
BASE + 8	b Routine3
BASE + C	b Routine4
BASE +10	•
BASE + n	•

```
INTR:•••

     Save state
     R3 <- @SIVEC
     R4 <-- Base of branch table

     •••

     lhz  RX, R3 (0)# load as ha
     add  RX, RX, R4
     mtsprCTR, RX
     bctr
```

BASE — 1st Instruction of Routir
BASE + 400 — 1st Instruction of Routir
BASE + 800 — 1st Instruction of Routir
BASE + C00 — 1st Instruction of Routir
BASE +1000
BASE + n

* **Note:** Interrupt code is defined such that its two least significant bits are zero, allowing indexing into the table.

Interrupt Context Switch

- After an interrupt (or any other exception) occurs, hardware:
  - Saves the state of the machine:

Copy **MSR** into **SRR1**

Loads **SRR0** with next Instruction Address or the instruction that caused the exception.

  - Changes the **MSR** including:

Switches to Supervisor mode, disables all maskable exceptions [EE bit = 0], and clears the recoverable interrupt(RI) bit

  - Branches to the exception vector (interrupt is **500**)

Point to the Exception vector and begin the service routine

- System issues user should consider
  - When to enable interrupts again
  - A non-maskable interrupt can come at any time

Interrupt Service Routine Steps (1 of 2)

1. Read the interrupt code in the SI Vector register (**SIVEC**) and go to service routine for that code.

2. If the interrupt source is IRQx edge-triggered, clear the service bit in the SI Pending Register(**SIPEND**). Bit is cleared by writing a one to it.

3. Save the SI Mask Register (**SIMASK**), and mask lower interrupt levels.
   - Required only if service routine is to be recoverable and lower priority interrupts are to be masked

4. In case a non-maskable exception occurs, save (**SRR0, SRR1**) on the stack, only if you want service routine to be recoverable.

Interrupt Service Routine Steps (2 of 2)

5. Set Recoverable Interrupt (**RI**) bit in MSR
   - Now that the previous state is saved, if we get a non-maskable interrupt we can recover
   - Load the External Interrupt Disable (**EID**) SPR
     - » sets the RI bit in the MSR
     - » keeps external interrupts disabled
     - » single instruction to set bit, e.g.,

       mtspr EID, gprx

6. Index to a branch table of Interrupt Handler Routines

7. Perform interrupt handler functions

8. Before the end of the routine
   - Restore the gpr(s) previously used
   - If a non-maskable exception occured during this interrupt service routine,
     - » Disable the Recoverable Interrupt (RI) bit in the MSR
       - Load any data to the NRI SPR, e.g.,

       mtspr NRI, gprx
     - » Restore SRR0 & SRR1 (and stack pointer if changed)
   - Execute “rfi” Instruction to return to application
     - » External interrupts will now be enabled again

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

Title: **irq1.c**
Handling an 860 SIU Interrupt

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

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:

irq1.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-D25 of ADS to J4-11 of UDLP1                */
/* (IRQ1.C)                                                   */

#include "mpc860.h"                 /* DUAL PORT RAM EQUATES */
struct dprbase *pdpr;               /* PNTR TO DUAL PORT RAM */

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

   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->PDDAT = 0;                 /* CLEAR PORT D DATA REG */
   pdpr->PDDIR = 0xff;              /* MAKE PORT D8-15 OUTPUT*/
   pdpr->SIEL.ED1 = 1;              /* MAKE IRQ1 FALLING EDGE*/
   pdpr->SIMASK.ASTRUCT.IRM1 = 1;   /* ENABLE IRQ1 INTERRUPTS*/
   asm(" mtspr 80,0");              /* ENABLE INTERRUPTS     */
   while (1==1);
}

#pragma interrupt intbrn
void intbrn()
{
   void irq1esr();

   asm (" stwu r9,-4(r1)");        /* PUSH GPR9 ONTO STACK  */
   switch (pdpr->SIVEC.IC)         /* PROCESS INTERRUPT CODE*/
   {
      case 8: asm (" mfspr r9,8"); /* PUSH LR ONTO STACK    */
              asm (" stwu r9,-4(r1)");
              asm (" bla irq1esr"); /* 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:;
   }
   asm (" lwz r9,0(r1)");          /* PULL GPR9 FROM STACK  */
   asm (" addi r1,r1,4");          /* RESTORE STACK POINTER */
}
```

irq1.c (2 of 2)

```
void irq1esr()
{
   pdpr->SIPEND = 1<<(31-2);         /* CLEAR IRQ1 INT PENDING*/
   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      */
   pdpr->PDDAT += 1;                 /* INCREMENT DISPLAY     */
   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");
}

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 */
}
```

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

Title: **irq0.c**
Handling 860 SIU Interrupt 0

Creation Date: **April 3, 1996** From: **IRQ1.C**

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, P10, A25 is connected to J4-11 on the UDLP1.**

Updates:

irq0.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-A25 of ADS to J4-11 of UDLP1                */
/* (IRQ0.C)                                                   */

#include "mpc860.h"                 /* DUAL PORT RAM EQUATES */
struct dprbase *pdpr;               /* PNTR TO DUAL PORT RAM */

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

   pdpr = (struct dprbase *) (getimmr() & 0xFFFF0000);
                                    /* INIT PNTR TO DPRBASE  */
   ptrs = (int *) intbrn;           /* INIT SOURCE POINTER   */
   ptrd = (int *)(getevt() + 0x100); /* 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*/
   pdpr->SIEL.ED0 = 1;              /* MAKE IRQ0 FALLING EDGE*/
   pdpr->SIMASK.ASTRUCT.IRM0 = 1;   /* ENABLE IRQ0 INTERRUPTS*/
   while (1==1);
}

#pragma interrupt intbrn
void intbrn()
{
   void irq0esr();

   asm (" stwu r9,-4(r1)");        /* PUSH GPR9 ONTO STACK  */
   switch (pdpr->SIVEC.IC)         /* PROCESS INTERRUPT CODE*/
   {
      case 0: asm (" mfspr r9,8"); /* PUSH LR ONTO STACK    */
              asm (" stwu r9,-4(r1)");
              asm (" bla irq0esr"); /* 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:;
   }
   asm (" lwz r9,0(r1)");          /* PULL GPR9 FROM STACK  */
   asm (" addi r1,r1,4");          /* RESTORE STACK POINTER */
}

void irq0esr()
{
   pdpr->SIPEND = 1<<(31-0);        /* CLEAR IRQ0 INT PENDING*/
   pdpr->PDDAT += 1;                /* INCREMENT DISPLAY     */
}
```

irq0.c (2 of 2)

```
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 */
}
```