

EEM489 MICROPROCESSORS II M68K LAB 5

1. Introduction

In this lab, you will write an Interrupt Service Routine in C language. PIT system's timer component will be utilized to light up and down a LED with a frequency of 10 Hz.

1.1. GNU C Support For Interrupt Service Routine

GNU C compiler supports writing interrupt service routines in a way very similar to writing function. The only difference is the prototype of the function. The prototype must be defined as follow:

```
void isr_timer(void) __attribute__((interrupt));

void main()
{

}

void isr_timer(void)
{
    int a;

    .....
}
```

GNU C compiler will substitute RTS with RTE to make sure the proper return from exception.

1.2. Inline Assembly

Immediately after interrupt service routine, we need to save all the registers. One alternative is to insert lines into the assembly output of the compiler. However, this is not simple. We should avoid modifying assembly outputs. The best way is to use inline assembly.

Assembly lines can be inserted from C source file with `asm(" ")` instruction. The only difference between `asm` instruction and regular assembly is in the definition of registers. All register names must start % (percent) sign. One example is as follows:

```
void isr_timer(void)
{
    asm("movem.l %a0-%a6/%d0-%d7, -(%sp)");

    .....

    asm("movem.l (%sp)+, %a0-%a6/%d0-%d7");
}
```

1.3. Frame Pointer

Well, if you compile the above code, you will get the following assembler output. Register A6 is used a frame pointer.

```
.globl isr_timer
.type      isr_timer,@function
hakan:
link.w    %a6,#0
#APP
movem.l   %a0-%a6/%d0-%d7,-(%sp)

movem.l   (%sp)+,%a0-%a6/%d0-%d7
#NO_APP
.L4:
unlk     %a6
rte
```

Frame pointer is a problem in interrupt service routines. Before saving registers, with `movem.l` instruction, frame pointer, a6 is updated. How can we omit frame pointer only for this function?

We need to transfer the interrupt service routine to a separate C source file, say `isr.c` and compile with the following command line.

```
...gcc -m68000 -Wall -fomit-frame-pointer -S -o isr.c isr.s
```

It produces assembly output by omitting frame pointer. An assembly output is obtained as follows:

```
.globl isr_timer
.type      isr_timer,@function
hakan:
#APP
movem.l   %a0-%a6/%d0-%d7,-(%sp)

movem.l   (%sp)+,%a0-%a6/%d0-%d7
#NO_APP
.L4:
rte
```

1.4. Accessing Device Registers

In order to be able to set up PI/T, we need to Access several device registers. These registers can be altered using pointers. For instance Counter Preload Register can be accessed as follows:

```
unsigned long *cpr = (unsigned long*)0xa00025;
unsigned char *tivr = (unsigned char*)0xa00023;
unsigned long *cntr = (unsigned long*)0xa0002d;
```

The rest of the registers can be created similarly. Altering the register content can be accomplished by pointers, e.g.

```
*cpr = 0x0010ffff;  
*tivr = 65;
```

1.5. Setting up Interrupts

As in the case of accessing device registers, you can set up the interrupt system. Changing interrupt vector can be done in a similar fashion. You are recommended to make a search to find examples of programmable interrupt timer usage in 68K. You can check Freescale forums.

2. Lab Work

- 2.1. Write an interrupt service routine (as a separate file) that is activated every 10 milliseconds producing a square wave from a port pin. Convert to assembly and check out the code.
- 2.2. Write the main program (as a separate file) that sets up the interrupt vectors and PI/T interrupts.
- 2.3. Link all the files and form a single S-file. Load the program onto lab board and run.
- 2.4. Check out the output. You can check out the output through application board that is used in eem336. Connect it to the 68k board using suitable links and try to switch the leds on application board. You need to soft-map (software mapping) 68k boards' pins to the application boards' pins. (Refer to the manuals of each board)