```
#include "armtimer.h"
#include "gpio.h"
#include "interrupts.h"
#include "pi.h"
#include "uart.h"
#include "printf.h"

// should not get called.
void impossible_handler(unsigned pc)
{
        printf("impossible exception at pc=%x\n", pc);
        pi_reboot();
}

volatile int counter = 0;

/*
 * we have only enabled timer interrupts
 */
void interrupt_handler(unsigned pc)
{
    armtimer_clear_interrupt();
    counter++;
}

void main(void)
{
    gpio_init();
    gpio_set_output(PI_ACT_LED);

        uart_init();
    printf("Executing program '%s'\n", __FILE__);

    // number of usecs between interrupts
        armtimer_init(1000000); // 1s
    armtimer_enable();      // enable timer

    armtimer_enable_interrupt();
    interrupts_enable_basic(INTERRUPTS_BASIC_ARM_TIMER_IRQ);
    system_enable_interrupts();

        int last = 0;
        while(1) {
                if (last != counter) {
                        last = counter;
            pi_led_toggle(PI_ACT_LED);
                        printf("received %d interrupts\n", last);
```

```
            }
        }

        pi_reboot();
}
```

```
/*
 * Reference full implementation of C start sequence for CS107E. This
 * function is called from start.s. _cstart() zeroes out the BSS
 * (assignment 4) and installs interrupt vectors (assignment 7). If main()
 * returns, it turns on the green ACT LED on the Raspberry Pi board.
 *
 * Author: Philip Levis <pal@cs.stanford.edu>
 * Author: Pat Hanrahan <hanrahan@cs.stanford.edu>
 * Author: Julie Zelenski <zelenski@cs.stanford.edu>
 *
 * Date: 6/20/17
 */

// linker memmap places bss symbols at start/end of bss
extern int __bss_start__, __bss_end__;

// _vector and _vector_end are symbols defined in the interrupt
// assembly file, at the beginning and end of the vector and its embedded constants
extern int _vectors, _vectors_end;

extern void main();

// The C function _cstart is called from the assembly in start.s
// _cstart zeroes out the BSS section and then calls main.
// After main() completes, turns on the green ACT LED as
// a sign of successful execution.
void _cstart() {
    int *bss = &__bss_start__;
    int *bss_end = &__bss_end__;

    while (bss < bss_end) {
        *bss++ = 0;
    }

    static int * const RPI_INTERRUPT_VECTOR_BASE = 0x0;

    /* Copy in interrupt vector table and FIQ handler at end of table. */
    int* vectorsdst = RPI_INTERRUPT_VECTOR_BASE;
    int* vectors = &_vectors;
    int* vectors_end = &_vectors_end;
    while (vectors < vectors_end) {
        *vectorsdst++ = *vectors++;
    }

    main();

    // Turn on the green ACT LED (GPIO 47)
```

```
    volatile unsigned int *GPIO_FSEL4  = (unsigned int *)0x20200010;
    volatile unsigned int *GPIO_SET1 =   (unsigned int *)0x20200020;
    *GPIO_FSEL4 = (1 << ((47-40)*3));
    *GPIO_SET1 = (1 << (47-32));
}
```

```
/*
 * Interrupt vectors.
 */
.globl _vectors
.globl _vectors_end

_vectors:
  ldr pc, _reset_asm
  ldr pc, _undefined_instruction_asm
  ldr pc, _software_interrupt_asm
  ldr pc, _prefetch_abort_asm
  ldr pc, _data_abort_asm
  ldr pc, _reset_asm
  ldr pc, _interrupt_asm
  ldr pc, _fast_asm

_reset_asm:                 .word impossible_asm
_undefined_instruction_asm:   .word impossible_asm
_software_interrupt_asm:      .word impossible_asm
_prefetch_abort_asm:          .word impossible_asm
_data_abort_asm:              .word impossible_asm
_interrupt_asm:               .word interrupt_asm
_fast_asm:                   .word impossible_asm

_vectors_end:

interrupt_asm:
       mov sp, #0x8000 @ Set interrupt stack pointer
    sub lr, lr, #4  @ Calculate pc to return to

    push {r0-r12,lr} @ Save all registers

    mov r0, lr       @ Call C interrupt handler with arg=pc
    bl interrupt_handler

    pop {r0-r12, lr} @ Restore registers
    movs pc, lr      @ Special return from interrupt
    @ movs restores processor mode

impossible_asm:
    mov sp, #0x7000  @ Set interrupt stack pointer
    sub lr, lr, #4
    bl impossible_handler    @ C function
    movs pc, lr
```