From $CS107E/src/interrupts_asm.s

```
interrupt_asm:
    mov   sp, #0x8000              @ init stack  (A)
    sub   lr, lr, #4               @ compute return addr
    push  {r0-r12, lr}             @ save all registers (B)
    mov   r0, lr                   @ pass old pc as arg
    bl    interrupt_dispatch       @ call C function
    ldm   sp!, {r0-r12, pc}^       @ resume, ^ changes mode
                                     & restore cpsr
```

(A) Could init fp to 0 or leave fp as-is. Affects end of backtrace from perspective of interrupt handler

(B) If assured that called routines obey C calling conventions, could save just callee-owned registers but for simplicity and safety, save all

From $CS107E/src/interrupts.c

```
void interrupts_attach_handler(handler_fn_t fn,
                               unsigned int source)
{
    if (!is_basic(source) && !is_shared(source)) return;

    assert(vector_table_is_installed());
    assert(nHandlers < MAX_HANDLERS);

    handlers[nHandlers].irq_source = source;
    handlers[nHandlers].fn = fn;
    nHandlers++;
    enable_source(source);
}
```

From $CS107E/src/interrupts.c

```
// check each handler for source, give a chance to process
// if handler returns true, indicates interrupt has
// been processed; dispatch stops here
// otherwise ask other attached handlers to process


void interrupt_dispatch(unsigned int pc)
{
    for (int i = 0; i < nHandlers; i++) {
       if (is_pending(handlers[i].irq_source)
                   && handlers[i].fn(pc)) {
            return;
        }
    }
}
```

Sample client use of interrupts module

```
   // interrupt handler function
bool button_pressed(unsigned int pc)
{
    if (gpio_check_and_clear_event(BUTTON)) {
        printf("+");
        return true;
    }
    return false;
}


void main(void)
{
    interrupts_init();
    gpio_enable_event_detection(BUTTON, GPIO_DETECT_FALLING_EDGE);
    interrupts_attach_handler(button_pressed, INTERRUPTS_GPIO3);
    interrupts_global_enable();
    ...
```