Interrupts

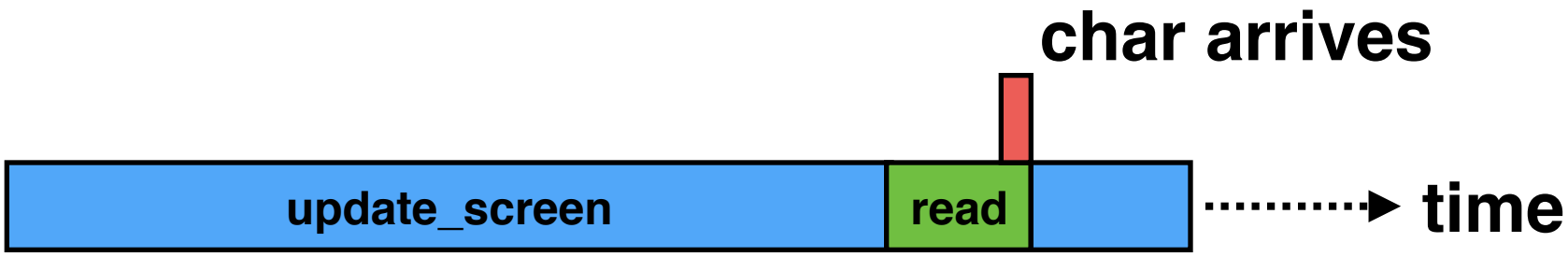
now, we're playing with gas

Blocking I/O

```
while (1) {
  read_char_to_screen();
  update_screen();
}
```

Blocking I/O

```
while (1) {
  read_char_to_screen();
  update_screen();
}
```

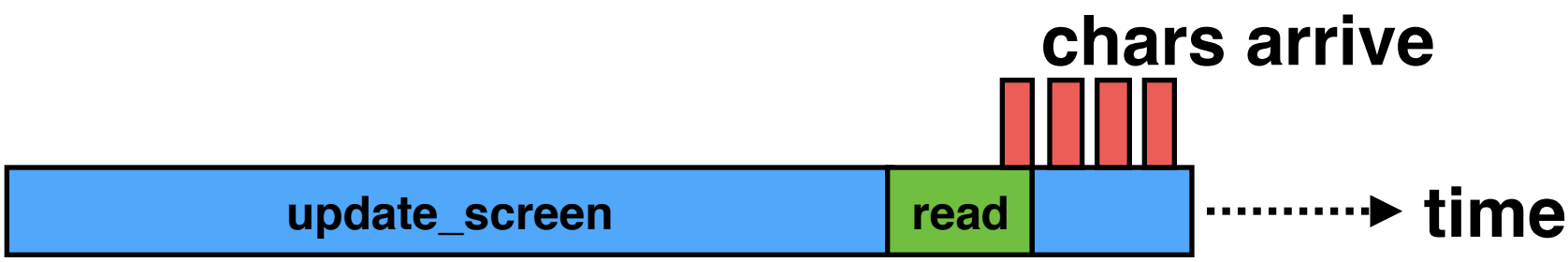


Blocking I/O

```
while (1) {
  read_char_to_screen();
  update_screen();
}
```

Blocking I/O

```
while (1) {
  read_char_to_screen();
  update_screen();
}
```

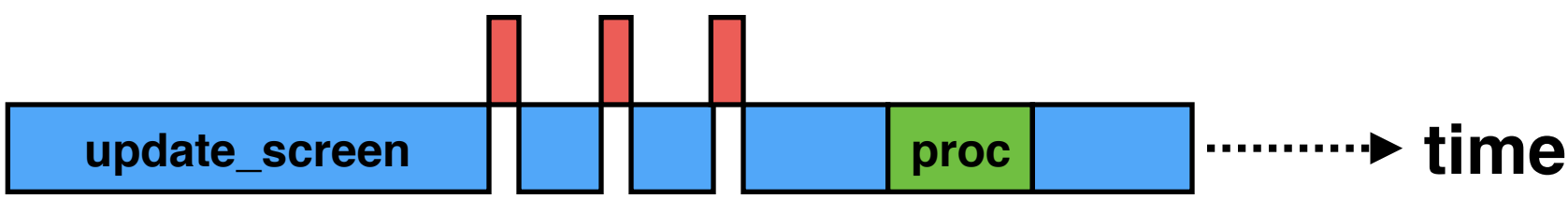


Button example

Concurrency

```
when a scan code arrives {
   add_scan_code_to_buffer();
}

while (1) {
   // Doesn't block
   process_scan_codes_to_screen();
   update_screen();
}
```



Interrupts

Events that cause processor to stop what it's doing and immediately execute other code, returning to original code when done.

- **External events (I/O, reset, timer)**
- **Internal events (bad memory access, software trigger).**

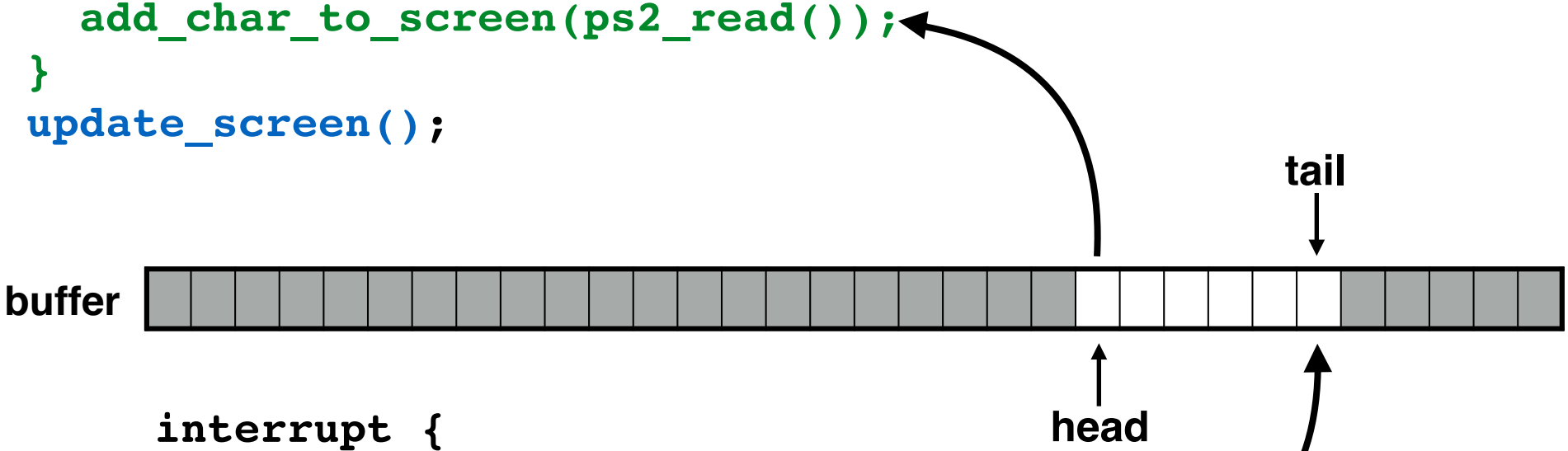
Critical for responsive systems

Using interrupts exercises everything you've learned so far

- **Architecture, assembly, loading, memory, C, peripherals**

PS2 Driver Software Model

```
while (1) {
  // Doesn't block
  while (ps2_has_chars()) {
    add_char_to_screen(ps2_read());
  }
  update_screen();
}
```




```
interrupt {
  read_data_bit();
  if (code_complete) {
    buffer_add(scan_to_ascii(code));
  }
}
```

Problem #1

```
Disassembly of section .text:

00008000 <_start>:
    8000:       e3a0d902        mov     sp, #32768      ; 0x8000
    8004:       eb000001        bl      8010 <_cstart>

00008008 <hang>:
    8008:       eb000039        bl      80f4 <led_on>
    800c:       eafffffe        b       800c <hang+0x4>

00008010 <_cstart>:
    8010:       e92d4800        push    {fp, lr}
```

Interrupt!

Processor Modes

Register	supervisor	interrupt
R0	R0	R0
R1	R1	R1
R2	R2	R2
R3	R3	R3
R4	R4	R4
R5	R5	R5
R6	R6	R6
R7	R7	R7
R8	R8	R8
R9	R9	R9
R10	R10	R10
fp	R11	R11
ip	R12	R12
sp	R13_svc	R13_irq
lr	R14_svc	R14_irq
pc	R15	R15
CPSR	CPSR	CPSR
SPSR	SPSR	SPSR

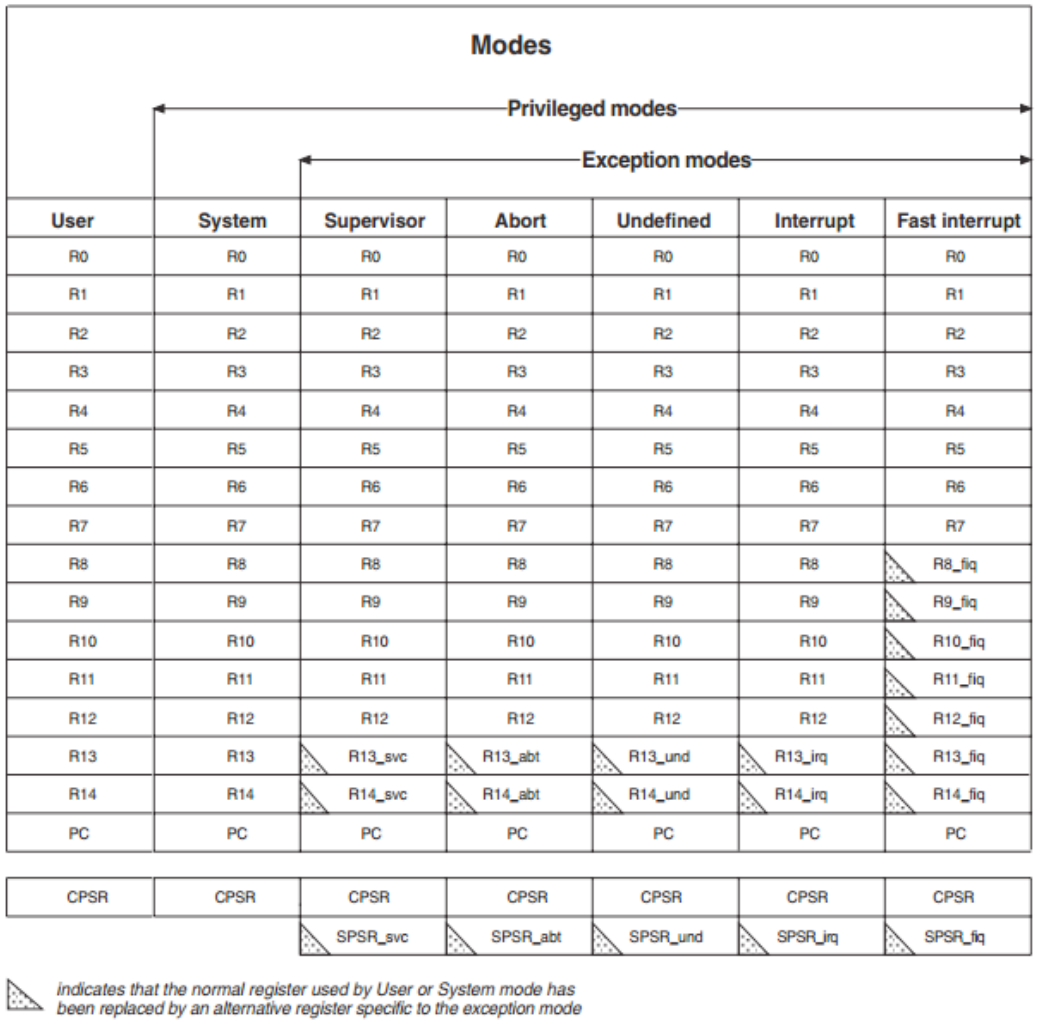
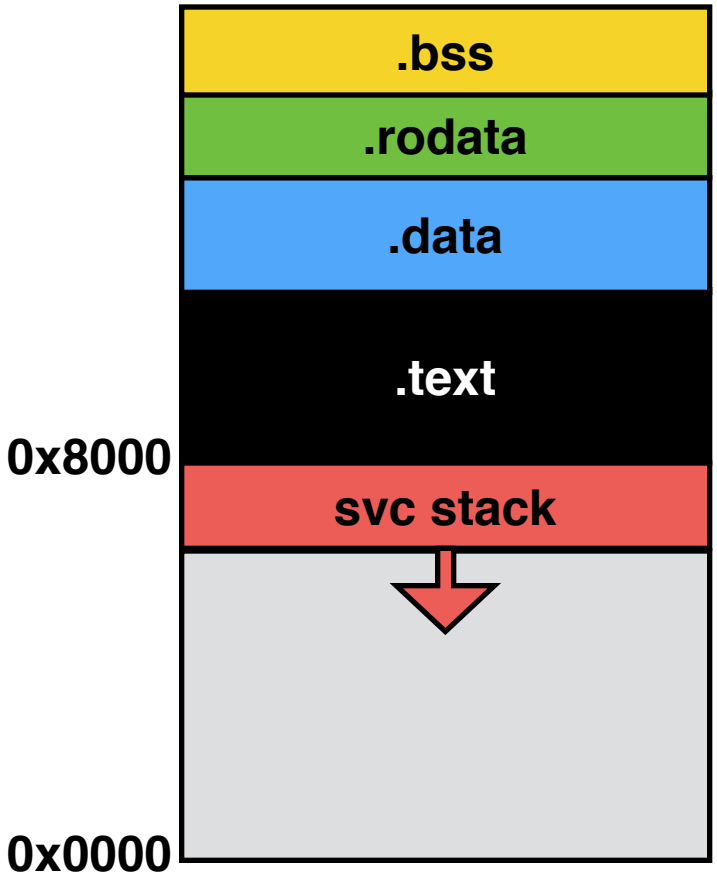
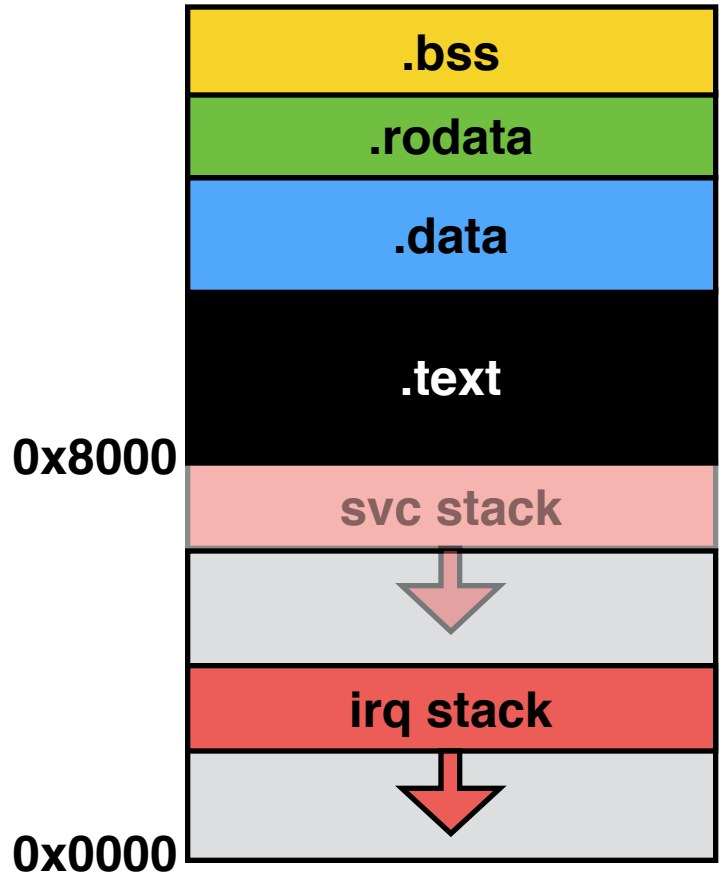


Figure A2-1 Register organization

Processor Modes, Cont'd

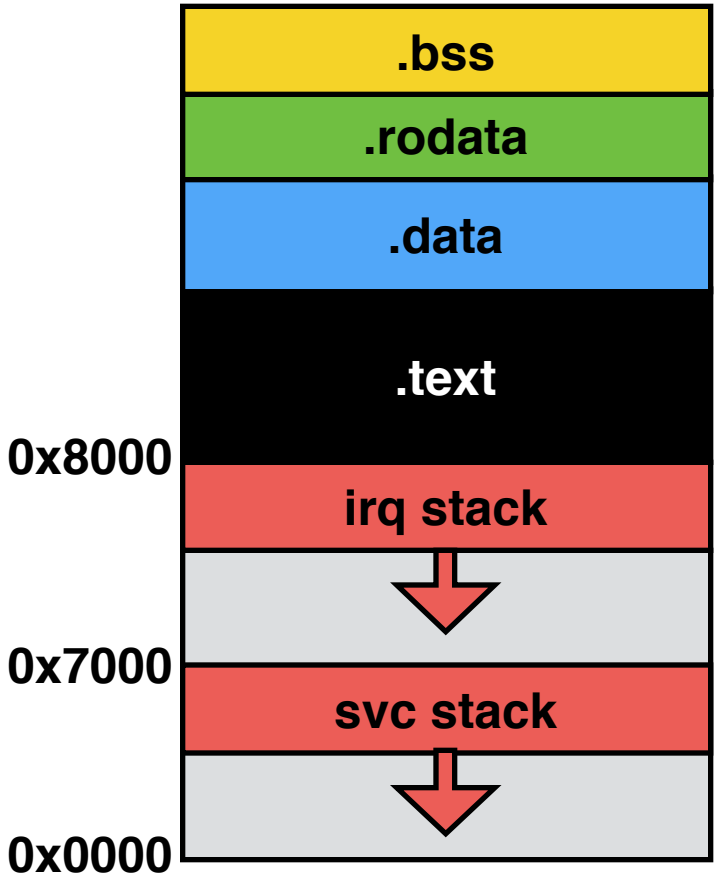


supervisor mode



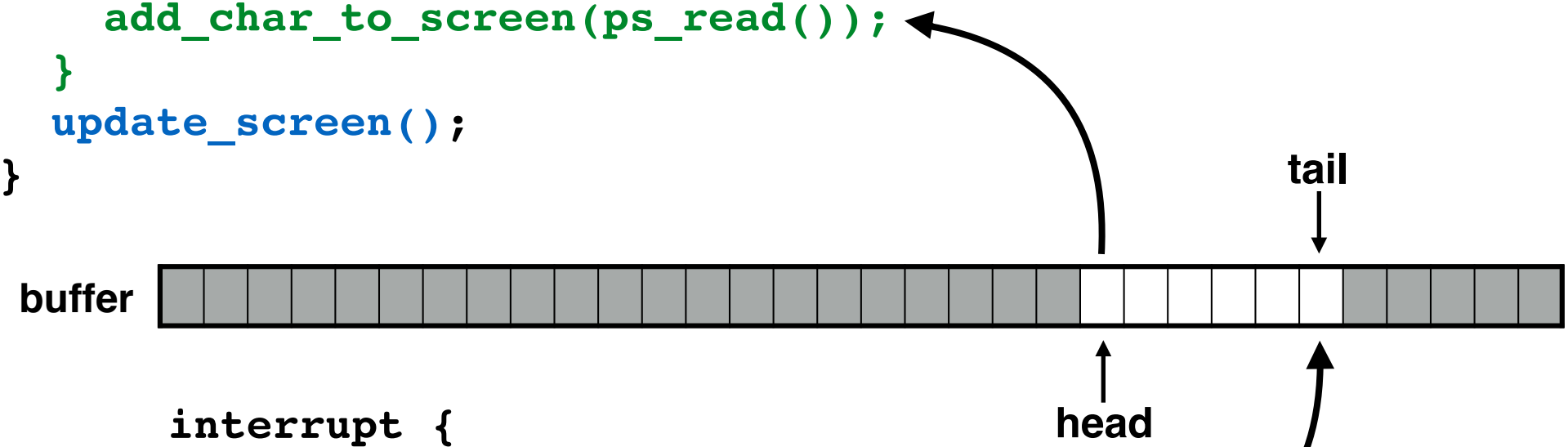
interrupt mode

Even Better Memory Layout



PS2 Driver Software Model

```
while (1) {
  // Doesn't block
  while (ps2_has_chars()) {
    add_char_to_screen(ps_read());
  }
  update_screen();
}
```




```
interrupt {
  read_data_bit();
  if (code_complete) {
    buffer_add(scan_to_ascii(code));
  }
}
```

We Need To

1. Set up the interrupt stack.

2. Write interrupt handler.
   - On clock from PS/2, read in bit, at end of byte put in buffer

3. Install interrupt handler code.

4. Tell CPU when to trigger interrupts.
   - When PS/2 clock line has a falling edge

5. Enable interrupts!

We Need To

1. Set up the interrupt stack.

2. Write interrupt handler.
   - On clock from PS/2, read in bit, at end of byte put in buffer

3. Install interrupt handler code.

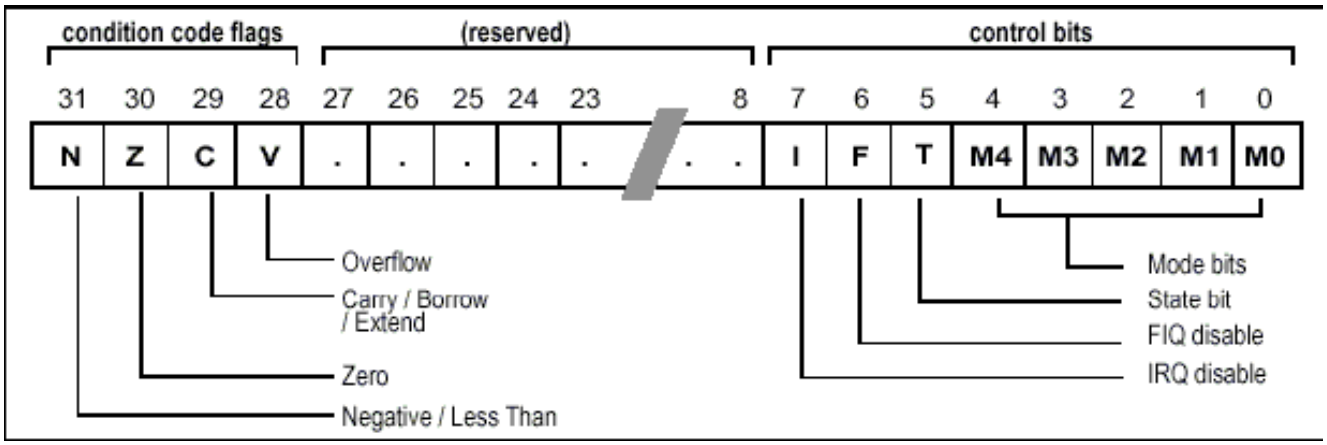
4. Tell CPU when to trigger interrupts.
   - When PS/2 clock line has a falling edge

5. Enable interrupts!

Setting up Interrupt Stack

1. Put processor into IRQ mode
2. Set stack pointer (r13)
3. Go back to SVC mode
4. Done!

CPSR



M[4:0]	Mode
b10000	User
b10001	FIQ
b10010	IRQ
b10011	Supervisor
b10111	Abort
b11011	Undefined
b11111	System

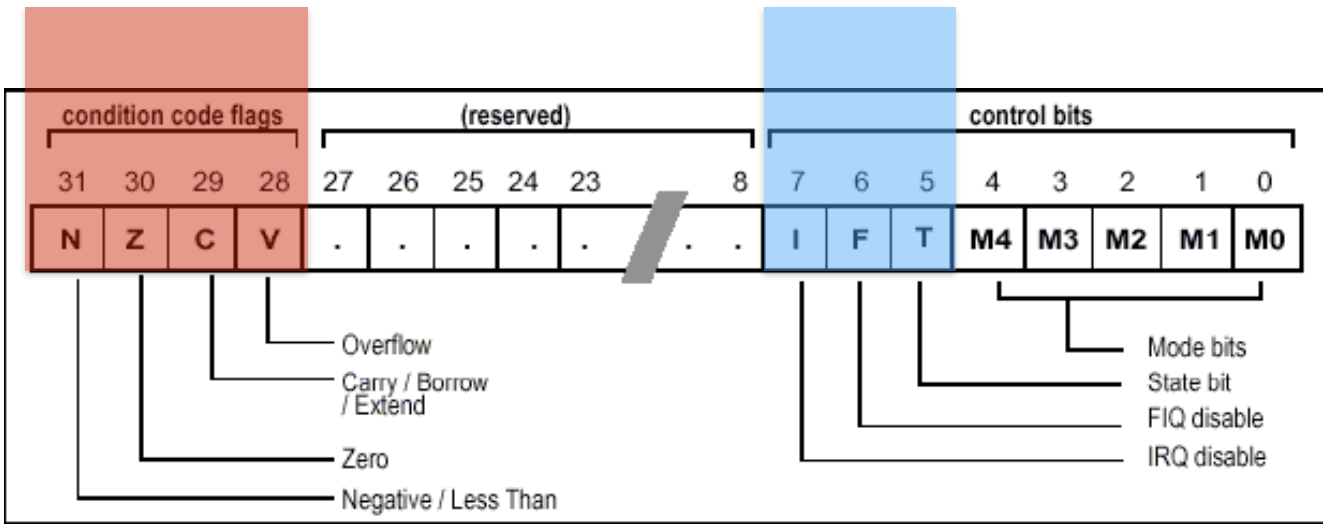
```
mrs      Rd, psr      <- Load Rd with psr
msr     psr, Rm       <- Store Rd into psr

mrs      r0, cpsr_c   <- Load r0 with CPSR
msr  cpsr_c, r0       <- Store CPSR with r0
```

CPSR

don't touch these bits (cpsr_c)

these bits should be 0b110



M[4:0]	Mode
b10000	User
b10001	FIQ
b10010	IRQ
b10011	Supervisor
b10111	Abort
b11011	Undefined
b11111	System

```
msr      psr, Rm      <- Store Rd into psr
mrs       Rd, psr     <- Load Rd with psr

msr  cpsr_c, r0       <- Store CPSR with r0
mrs       r0, cpsr_c  <- Load r0 with CPSR
```

We Need To

1. Set up the interrupt stack.

2. **Write interrupt handler.**
   - **On clock from PS/2, read in bit, at end of byte put in buffer**

3. Install interrupt handler code.

4. Tell CPU when to trigger interrupts.
   - When PS/2 clock line has a falling edge

5. Enable interrupts!

Interrupt Execution

```
00008000 <notmain>:
    ...
    816c:       e5cd900c        strb    r9, [sp, #12]
    8170:       e5cd800d        strb    r8, [sp, #13]
    8174:       e7d32006        ldrb    r2, [r3, r6]
    8178:       e28d300d        add     r3, sp, #13
    ...
    81bc:       eb0001b7        bl      88a0 <gfx_draw>   ←——— lr = 81c0
    81c0:       eb000454        bl      9318 <led_toggle>
    81c4:       e5973000        ldr     r3, [r7]
    81c8:       e3530063        cmp     r3, #99 ; 0x63
```

Interrupt Execution

```
00008000 <notmain>:
    ...
    816c:       e5cd900c        strb    r9, [sp, #12]
    8170:       e5cd800d        strb    r8, [sp, #13]   ← lr = 8174
    8174:       e7d32006        ldrb    r2, [r3, r6]
    8178:       e28d300d        add     r3, sp, #13
    ...
    81bc:       eb0001b7        bl      88a0 <gfx_draw>   ← lr = 81c0
    81c0:       eb000454        bl      9318 <led_toggle>
    81c4:       e5973000        ldr     r3, [r7]
    81c8:       e3530063        cmp     r3, #99 ; 0x63
```

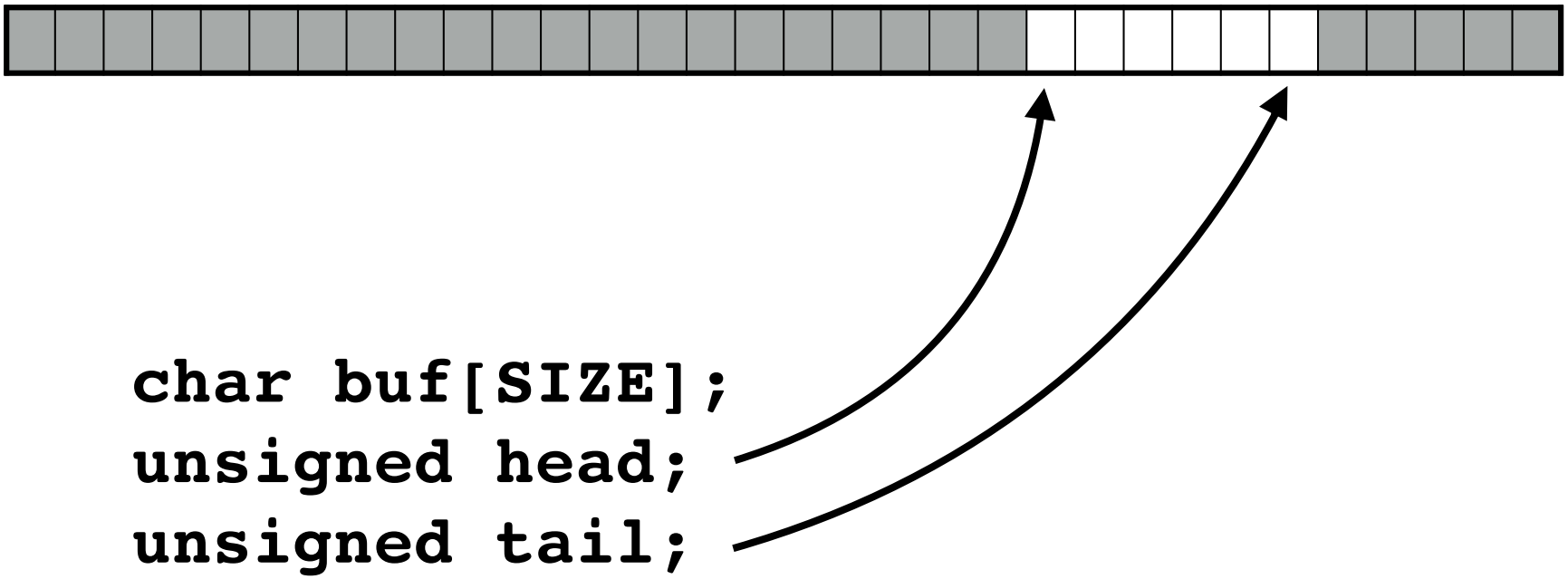
If we use a normal function, the interrupt will skip an instruction when it returns.

Interrupt Code

```
interrupt_asm:
  sub lr, lr, #4
  push {lr}
  push {r0-r12}
  mov r0, lr                @ Pass old pc
  bl interrupt_vector       @ C function
  pop {r0-r12}
  ldm sp!, {pc}^

void interrupt_vector(unsigned pc) {
  // Read in bit from GPIO24
  // Process bit of PS2 packet
  // If byte complete, insert in buffer
}
```

Buffer



```
char buf[SIZE];
unsigned head;
unsigned tail;
```

```
int insert(char ch) {
  if (!((tail + 1) % SIZE) == head) {
    tail++;
    buf[tail] = ch;
    return 1;
  } else {
    return 0;
  }
}
```

We Need To

1. Set up the interrupt stack.

2. Write interrupt handler.
   - On clock from PS/2, read in bit, at end of byte put in buffer

3. Install interrupt handler code.

4. Tell CPU when to trigger interrupts.
   - When PS/2 clock line has a falling edge

5. Enable interrupts!

ARMv6 Interrupts

Normal Address	Exception	Mode
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt (SWI)	Supervisor
0x0000000C	Prefetch Abort	Abort
0x00000010	Data Abort	Abort
0x00000018	IRQ (Interrupt)	IRQ
0x0000001C	FIQ (Fast Interrupt)	IRQ

Desired Assembly

```
00000000:
    0: ldr pc, =reset_asm
    4: ldr pc, =undefined_instruction_asm
    8: ldr pc, =software_interrupt_asm
    c: ldr pc, =prefetch_abort_asm
   10: ldr pc, =data_abort_asm
   14: ldr pc, =reset_asm
   18: ldr pc, =interrupt_asm
   1c: ldr pc, =fast_interrupt_asm
```

Generate this assembly code and copy it to exception table location (0x00000000).

Generating Assembly

```
.globl _table

_table:
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =reset
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
```

```
0000849c <_table>:
    849c:       e59ff018        ldr     pc, [pc, #24]   ; 84bc <_table+0x20>
    84a0:       e59ff014        ldr     pc, [pc, #20]   ; 84bc <_table+0x24>
    84a4:       e59ff010        ldr     pc, [pc, #16]   ; 84bc <_table+0x24>
    84a8:       e59ff00c        ldr     pc, [pc, #12]   ; 84bc <_table+0x24>
    84ac:       e59ff00c        ldr     pc, [pc, #12]   ; 84c0 <_table+0x20>
    84b0:       e59ff008        ldr     pc, [pc, #8]    ; 84c0 <_table+0x20>
    84b4:       e51ff000        ldr     pc, [pc, #-0]   ; 84bc <_table+0x24>
    84b8:       e51ff004        ldr     pc, [pc, #-4]   ; 84bc <_table+0x24>
    84bc:       000096c0        .word   0x000096c0
    84c0:       00008290        .word   0x00008290
```

Generating Assembly

```
.globl _table

_table:
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =reset
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
```

These constants could end up anywhere.

```
0000849c <_table>:
    849c:       e59ff018        ldr     pc, [pc, #28]   ; 84c0 <_table+0x24>
    84a0:       e59ff014        ldr     pc, [pc, #20]   ; 84bc <_table+0x20>
    84a4:       e59ff010        ldr     pc, [pc, #16]   ; 84bc <_table+0x20>
    84a8:       e59ff00c        ldr     pc, [pc, #12]   ; 84bc <_table+0x20>
    84ac:       e59ff00c        ldr     pc, [pc, #12]   ; 84c0 <_table+0x24>
    84b0:       e59ff008        ldr     pc, [pc, #8]    ; 84c0 <_table+0x24>
    84b4:       e51ff000        ldr     pc, [pc, #-0]   ; 84bc <_table+0x20>
    84b8:       e51ff004        ldr     pc, [pc, #-4]   ; 84bc <_table+0x20>
    84bc:       000096c0        .word   0x000096c0
    84c0:       00008290        .word   0x00008290
```

Generating Assembly

```
.globl _table

_table:
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =reset
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
```

These constants could end up anywhere.

What's funny here?

```
0000849c <_table>:
    849c:       e59ff018        ldr     pc, [pc, #28]   ; 84c0 <_table+0x24>
    84a0:       e59ff014        ldr     pc, [pc, #20]   ; 84bc <_table+0x20>
    84a4:       e59ff010        ldr     pc, [pc, #16]   ; 84bc <_table+0x20>
    84a8:       e59ff00c        ldr     pc, [pc, #12]   ; 84bc <_table+0x20>
    84ac:       e59ff00c        ldr     pc, [pc, #12]   ; 84c0 <_table+0x24>
    84b0:       e59ff008        ldr     pc, [pc, #8]    ; 84c0 <_table+0x24>
    84b4:       e51ff000        ldr     pc, [pc, #-0]   ; 84bc <_table+0x20>
    84b8:       e51ff004        ldr     pc, [pc, #-4]   ; 84bc <_table+0x20>
    84bc:       000096c0        .word   0x000096c0
    84c0:       00008290        .word   0x00008290
```

Explicit Embedding

```
.globl _table

_table:
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
ldr pc, =reset
ldr pc, =reset
ldr pc, =interrupt_asm
ldr pc, =interrupt_asm
```

```
.globl _table

_table:
ldr pc, _reset
ldr pc, _interrupt
ldr pc, _interrupt
ldr pc, _interrupt
ldr pc, _reset
ldr pc, _reset
ldr pc, _interrupt
ldr pc, _interrupt

_interrupt:  .word interrupt_asm
_reset:      .word reset
```

Now we know the constants will follow the code.

C Code

```
extern unsigned int _table[];
#define INTERRUPT_TABLE_SIZE 10
#define RPI_INTERRUPT_VECTOR_BASE 0x0

unsigned int i;
for (i = 0; i < INTERRUPT_TABLE_SIZE; i++) {
  ((unsigned int*)RPI_INTERRUPT_VECTOR_BASE)[i] = _table[i];
}
```

```
0000849c <_table>:
    849c:       e59ff018        ldr     pc, [pc, #28]   ; 84c0 <_table+0x24>
    84a0:       e59ff014        ldr     pc, [pc, #20]   ; 84bc <_table+0x20>
    84a4:       e59ff010        ldr     pc, [pc, #16]   ; 84bc <_table+0x20>
    84a8:       e59ff00c        ldr     pc, [pc, #12]   ; 84bc <_table+0x20>
    84ac:       e59ff00c        ldr     pc, [pc, #12]   ; 84c0 <_table+0x24>
    84b0:       e59ff008        ldr     pc, [pc, #8]    ; 84c0 <_table+0x24>
    84b4:       e51ff000        ldr     pc, [pc, #-0]   ; 84bc <_table+0x20>
    84b8:       e51ff004        ldr     pc, [pc, #-4]   ; 84bc <_table+0x20>
    84bc:       000096c0        .word   0x000096c0
    84c0:       00008290        .word   0x00008290
```

```
00000000:
    0000:       e59ff018        ldr     pc, [pc, #28]   ; 0028
    0004:       e59ff014        ldr     pc, [pc, #20]   ; 0020
    0008:       e59ff010        ldr     pc, [pc, #16]   ; 0020
    000c:       e59ff00c        ldr     pc, [pc, #12]   ; 0020
    0010:       e59ff00c        ldr     pc, [pc, #12]   ; 0024
    0014:       e59ff008        ldr     pc, [pc, #8]    ; 0024
    0018:       e51ff000        ldr     pc, [pc, #-0]   ; 0020
    001c:       e51ff004        ldr     pc, [pc, #-4]   ; 0020
    0020:       000096c0        .word   0x000096c0
    0024:       00008290        .word   0x00008290
```

We Need To

1. Set up the interrupt stack.

2. Write interrupt handler.
   - On clock from PS/2, read in bit, at end of byte put in buffer

3. Install interrupt handler code.

4. Tell CPU when to trigger interrupts.
   - When PS/2 clock line has a falling edge

5. Enable interrupts!

Interrupts Overview

Problem: responsive PS2 driver.

Answer: run interrupt code in response to events or inputs, CPU preempts execution, no blocking needed.

Requires setting up CPU to execute code, CPU provides some extra mechanisms and has different execution modes.

Executed code has some special properties.