Lecture 31

Interrupts

- CPU saves PC of interrupted instruction to uepc register
- CPU jumps to address of interrupt handler
- Executes handler code and then returns to interrupted instruction, uret copies uepc to PC

Enabling Interrupts in RISC-V

1. Configure device steps specific to device
2. Set utvec
3. Set interrupt enable but in ustatus reg
4. Set interrupt but in uie register

Example

- Timer Device

```
.data
curr_time: .word 0xFFFF0018
time_cmp: .word 0xFFFF0020
counter: .word 0
.text
.global_start
_start:
la s0, time_cmp
li s1, 1000 # 1000 ms
sw s1, 0(s0) # set timer value
# setup interrupts
la t0, timer_handler
csrrw zero, utvec, t0 # set utvec to handler addr
csrrsi zero, utstatus, 0x1 # set interrupt enable bit
csrrsi zero, ue, 0x10
LOOP: j LOOP # do sth while waiting for interrupt
timer_handler:
addi, sp, sp, -12
sw t0, 0(sp)
sw t1, 4(sp)
sw t2, 8(sp)
la t1, counter
lw t2, 0(t1)
addi, t2, t2, 1
sw t2, 0(t1)
# set new interrupt time
lw t0, 0(sp)
lw t1, 4(sp)
```

```
lw t2, 8(sp)
addi sp, sp, 12
uret # return to uepc-pc
```

Summary

- Configure the device
- Configure CPU, CSRs
- Write interrupt handler; be sure to save/restore any registers used in interrupt handler

Wrap up

- Polled I/O can be inefficient
- Interrupt driven I/O are better (but more annoying to code)

Timing Analysis

- Delays determine clock frequency of a circuit

```
module muxDFF(input logic w, r, L, E, clock, output logic Q);
    always_ff @(posedge clock)
        if(L)
            Q <= r;
        else if (E)
            Q <= w;
endmodule
module shift4(input logic[3:0] sw, KEY, output logic[3:0] LEDR);
    logic[3:0] Q;
    muxDFF u3(KEY[0], sw[3], KEY[2], KEY[1], KEY[3], Q[3]);
    muxDFF u2(Q[3], sw[2], KEY[2], KEY[1], KEY[3], Q[2]);
    muxDFF u1(Q[2], sw[1], KEY[2], KEY[1], KEY[3], Q[1]);
    muxDFF u0(Q[1], sw[0], KEY[2], KEY[1], KEY[3], Q[0]);
    assign LEDR = Q;
endmodule
```