Lecture 30

Polling

- Polling iteratively checks to see if device is ready
- I/O with keys and LEDRs

```
li s0, 0xFF200050 # address of keys in S0
li s1, 0xFF200000 # address of LEDRs in S1
POLL:   lw s2, 0(s0) # poll for key press
    beqz s2, POLL # branch to poll if no key pressed
WAIT:   lw s3, 0(s0)
    bnez s3, WAIT # wait for key release, as key is usually pressed for quite some clock cyc
    li s3, 1 # s2 will be 1 if key0 was pressed
    bne s2, s3, check_1
    li s4, 0 # value to be written to LEDs
    j update_led
check_1:li s3, 2 # 0b0010, key 1
    bne s2, s3, check_2
    li s4, 1 # value to be written to LEDs
    j update_led
check_2: li s3, 4 # 0b0100
    bne s2, s3, check_3
    li s4, 3 # 0b0011
    j update_led
check_3: li s4, 7 #0b0111
update_led: s2 s4, 0(s1) # lights up LEDs
j POLL
```

- Poll is called repeatedly to check/read I/O

Wrap-Up

- CPU can interact with devices just like with memory
- Polling can be used to check if devices are ready

Interrupts

- Polled I/O can be inefficient
    - prevents processor from doing other work
    - instead continue executing code and let device interrupt processor

1. CPU configures for interrupts
2. CPU does other work
3. CPU gets interrupted
    - CPU saves state so it knows where it is interrupted
    - CPU jumps to interrupt handler
    - CPU resumes what it was doing

- Polling vs Interrupts

	Polling	Interrupt
When	Explicit polling loop in code	Anytime
Difficulty	Easy	More complex
Efficiency	good if small wait is expected	good for medium to long wait

- Example events handled by interrupts
  - External devices: USB
  - Operating system: timers, disk I/O, keyboard I/O
  - Debugging: breakpoints
  - Program errors: divide by 0, misaligned memory access, memory protection violation (segfault)
- Exception vs Interrupt
  - Exception: internal, arise within CPU, program errors
  - Interrupt: external, comes from outside processor
  - Some use it interchangeably
- Simplified interrupt hardware
  - IRQ: interrupt request: device asserts to interrupt CPU
  - IACK: interrupt acknowledge
- Interrupt handling requires special system instructions and registers
  - Control and Status Registers (CSRs): special registers to monitor system state. Can be read, written, bits set, cleaned via special instructions. Each CSR has special name, unique function, such as ustatus, uie (interrupt enable), utvec: user trap handler base address.
- System instructions
  - ebreak: pause execution
  - uret: return from handling interrupt
  - csrrw: read/write CSR
  - csrrsi: read/set bits in CSR