Interrupts vs. Polling

Polling

- Constantly reading a memory location, in order receive updates of an input value

```c
#include "system.h"
#include "io.h"

int main() {
    unsigned char key;
    unsigned char val = 0x01;
    IOWR( LED_BASE, 0, val );
    while(1) {
        key=IORD(KEY_BASE,0);
        if( key == 0x02 ){
            // key number 1 pressed
            // slide the bit one position to the left
            val = val << 1 | val >> 7;

            // update the LEDs
            IOWR( LED_BASE, 0, val );
            // wait until the key is released
            while( IORD( KEY_BASE,0) != 0x03 ));
        }
    }
    return 0;
}
```

Polling

Polling

Interrupts

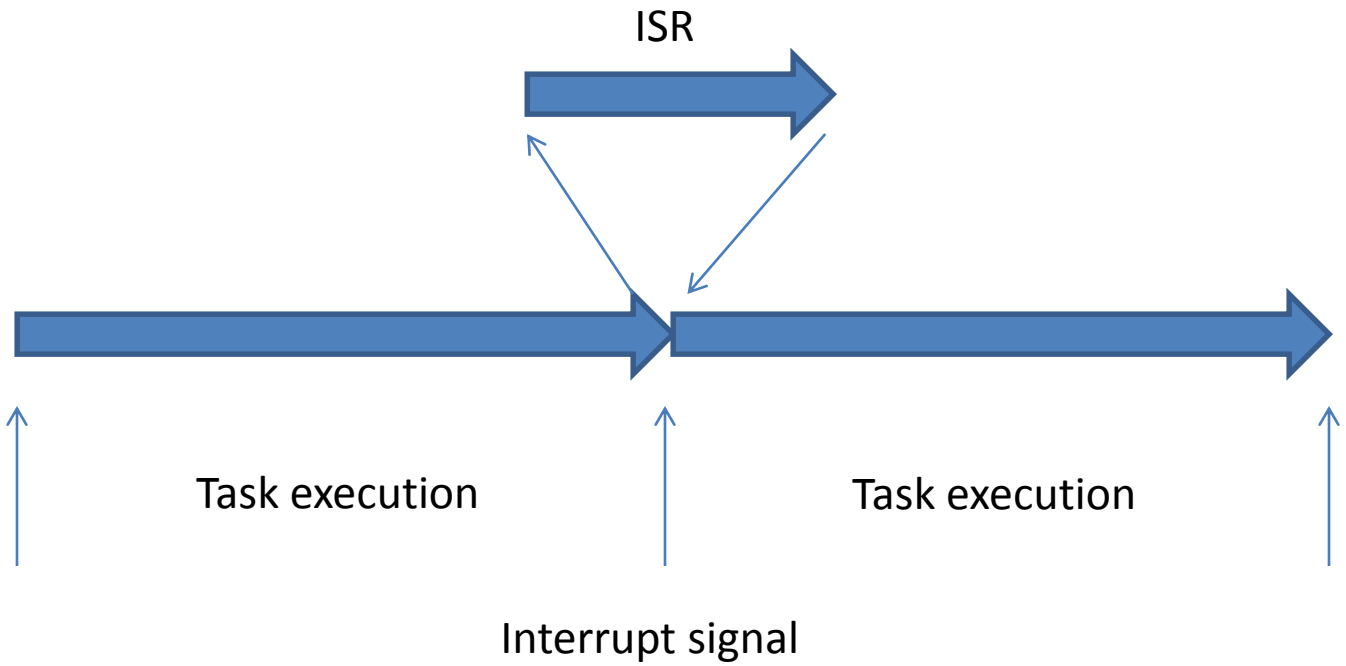
- "Normal" microcontroller flow of control
  - Single process
  - Main routine / main loop
  - Normal control flow statements (if, while, for, etc.)

- Asynchronous, important events
  - May need to be handled immediately
  - Realtime requirements
  - Interrupt normal flow of control

Interrupt Handling

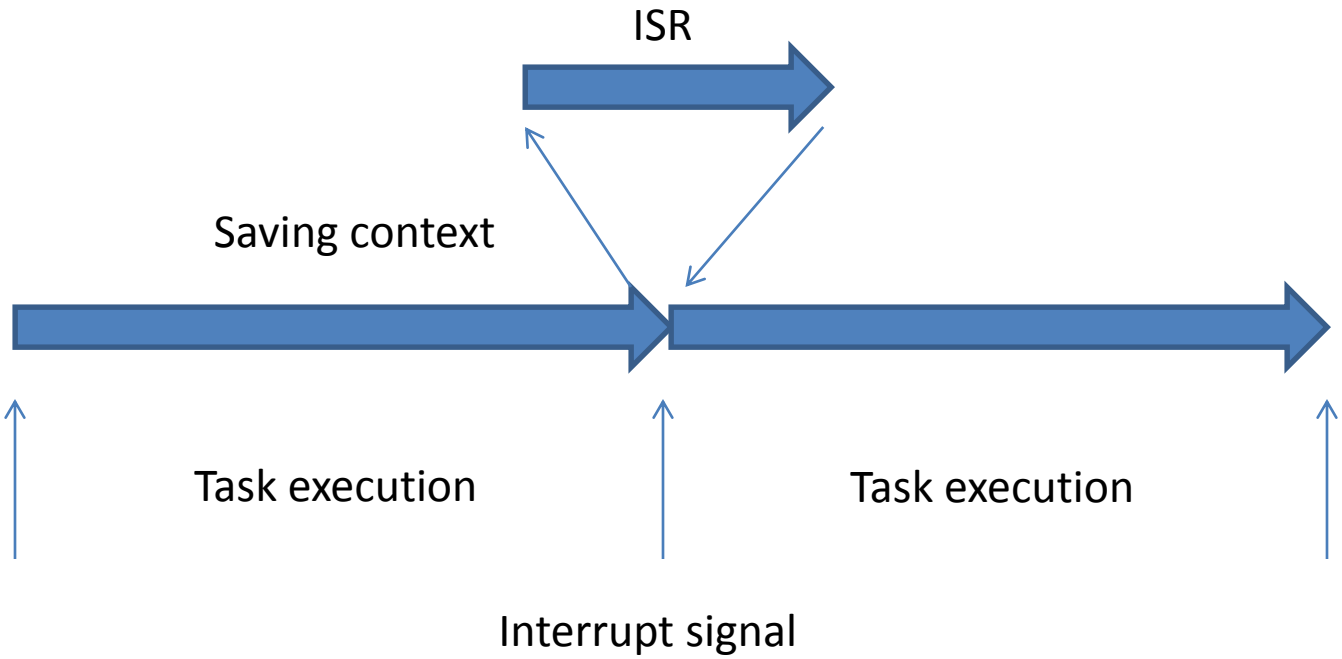
- Processor interrupts (preempts) the current flow of control
- Time spent in interrupt handlers should be kept as short as possible
- Microcontroller offers interrupts for various conditions
  - Not all are useful all the time: enable required interrupts
  - Some critical may require atomic execution (no interruptions guaranteed)
  - Disable / re-enable interrupts around critical section

In details



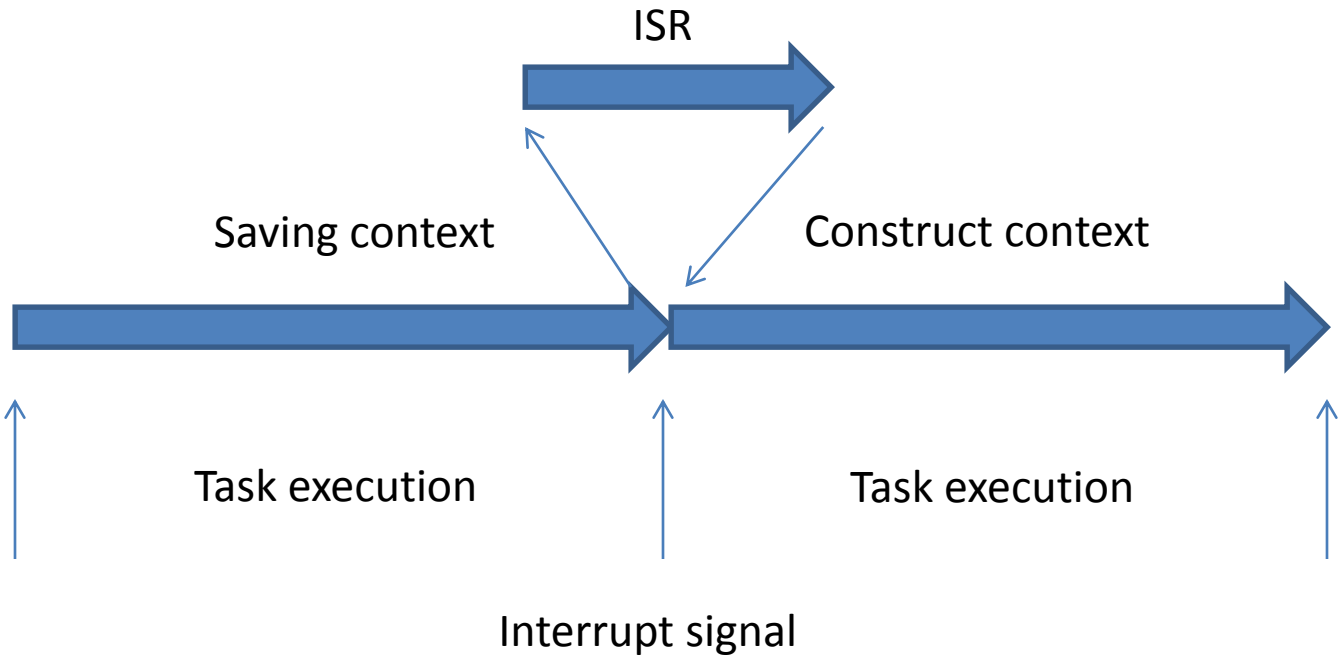
In details

- Saving context: Push all temporary variables (like program counter) into stack



In details

- Construct context: Pull all temporary variables (like program counter) out of stack



Sources of Interrupts

- Timers: System “ticks”, periodic tasks
- Communications
  - Ethernet
  - USB
  - Serial
- Periphery
  - E.g. ADC (Conversion complete)
  - Memory management
- Software

  Software interrupts (trap instructions) / illegal instructions
- Reset / Power-On

Interrupt Priorities

- What happens if a interrupt occurs during another interrupt is handled?
  - E.g. Timer ticks while Ethernet packet is processed
  - Priorities: Some interrupts are more important than other
  - Hierarchy of interrupts
    - Only more important interrupts may preempt lesser interrupts
    - Usually: lower priority is more important
    - Interrupts on same priority level do not preempt
  - Not immediately handled interrupts may be stored or may be lost
    - Depends if interrupt condition persists

Interrupt Latency

- Latency
  - Time passed between occurrence of event and handling
  - Interrupt is generated and source is serviced

- Interrupt latency
  - Interrupts tightly integrated into processor
  - Low latency
  - Complete current instruction, save registers and jump to handler
  - Measured in instruction cycles

Interrupts vs. Polling

- Polling:
  - Continuously poll IOs for change of value
  - Pro:
    - Short latencies (if low #IOs)
    - Events do not block the normal program exec.
  - Cons:
    - Most polls are unneeded – value did not change
    - High CPU usage
    - Reaction time depends on #IOs
- Interrupt
  - Normal execution is interrupted when event occurs
  - Pro:
    - Processor resources are only used when necessary
  - Cons:
    - Program execution is interrupted in a non-deterministic manner

Interrupt Service Routine (ISR)

- Event handler for interrupt

- Special, user-defined function for handling the interrupt

Trigger for external interrupts

- Edge-triggered
  - Falling Edge – when signal value *decreases*
  - Rising Edge – when signal value *increases*
  - Either Edge
- Level-triggered
  - each instance, when signal is above or below a certain threshold
- IRQs can be configured from within SOPC Builder

Altera Interrupts

- Initialization
- Enable the interrupt for the specific input

```
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(<BASE>, <MASK>);
```

- Set the edge capability

```
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(<BASE>, <VAL>)
```

- Register handler

```
alt_ic_isr_regsiter( <IRQ_Controller_ID>, <IRQ>,
                     <isr_function>, <isr_context>, <flags> )
```

Tasks

- Try out and understand the Interrupt based KEY-LED package
- Your code need to print “No Key is pressed!“ when no key was pressed. At the same time, you code need to reponse the key press event and turn on and off the led alternately.
- Tell what you see and explain it.