Today

- Exceptional Control Flow
- Exceptions
- Non-local jumps (setjmp/longjmp)

Control Flow

- Processors do only one thing:
  - From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
  - This sequence is the CPU's control flow (or flow of control)

  Physical control flow

```
<startup>
inst_1
inst_2
...
inst_n
<shutdown>
```

Alterating the Control Flow

- Up to now: two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return
  - React to changes in program state

- Insufficient for a useful system:
  - Difficult to react to changes in system state
    - Data arrives from a disk or a network adapter
    - Instruction divides by zero
    - User hits Ctrl-C at the keyboard
    - System timer expires

- System needs mechanisms for "exceptional control flow"

Exceptional Control Flow

- Exists at all levels of a computer system
- Low level mechanisms
  - 1. Exceptions
    - Change in control flow in response to a system event (i.e., change in system state)
    - Implemented using combination of hardware and OS software
  - Higher level mechanisms
    - 2. Process context switch
      - Implemented by OS software and hardware timer
    - 3. Signals
      - Implemented by OS software
    - 4. Nonlocal jumps: `setjmp()` and `longjmp()`
      - Implemented by C runtime library

Today

- Exceptional Control Flow
- Exceptions
- Non-local jumps (setjmp/longjmp)
### Exceptions

- **An exception** is a transfer of control to the OS kernel in response to some event (i.e., change in processor state).
- Kernel is the memory-resident part of the OS.
- Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C.

![User code vs Kernel code](image)

### Exception Tables

- Each type of event has a unique exception number $k$.
- $k$ is the index into the exception table (a.k.a. interrupt vector).
- Handler $k$ is called each time exception $k$ occurs.

### Asynchronous Exceptions (Interrupts)

- **Caused by events external to the processor**
  - Indicated by setting the processor's interrupt pin.
  - Handler returns to "next" instruction.
- **Examples**:
  - Timer interrupt
  - Every few ms, an external timer chip triggers an interrupt.
  - Used by the kernel to take back control from user programs.
  - I/O interrupt from external device.
  - Hitting Ctrl-C at the keyboard.
  - Arrival of a packet from a network.
  - Arrival of data from a disk.

### Synchronous Exceptions

- **Caused by events that occur as a result of executing an instruction**:
  - Traps
    - Intentional
    - Examples: `system calls`, breakpoint traps, special instructions.
    - Returns control to "next" instruction.
  - Faults
    - Unintentional but possibly recoverable.
    - Examples: page faults (recoverable), protection faults (unrecoverable), floating point exceptions.
    - Either re-executes faulting ("current") instruction or aborts.
  - Aborts
    - Unintentional and unrecoverable.
    - Examples: illegal instruction, parity error, machine check.
    - Aborts current program.

### Nonlocal Jumps: `setjmp/longjmp`

- **Powerful (but dangerous) user-level mechanism for transferring control to an arbitrary location**
  - Controlled to way to break the procedure call / return discipline.
  - Useful for error recovery and signal handling.
- **`int setjmp(jmp_buf j)`**
  - Must be called before `longjmp`.
  - Identifies a return site for a subsequent `longjmp`.
  - Called once, returns one or more times.
- **Implementation**:
  - Remember where you are by storing the current `register context`, `stack pointer`, and `PC value in jmp_buf`.
  - Return 0.

### Today

- Exceptional Control Flow
- Exceptions
- Non-local jumps (`setjmp/longjmp`)
setjmp/longjmp (cont)

- void longjmp(jmp_buf j, int i)
  - Meaning:
    - return from the setjmp remembered by jump buffer j again ...
    - ... this time returning i instead of 0
  - Called after setjmp
  - Called once, but never returns

- longjmp implementation:
  - Restore register context (stack pointer, base pointer, PC value) from jump buffer j
  - Set $%eax$ (the return value) to i
  - Jump to the location indicated by the PC stored in jump buffer j

setjmp/longjmp Example

- Goal: return directly to original caller from a deeply-nested function

```c
/* Deeply nested function foo */
void foo(void)
{
  if(error1)
    longjmp(buf, 1);
  bar();
}
void bar(void)
{
  if(error2)
    longjmp(buf, 2);
}
```

Limitations of Nonlocal Jumps (cont.)

- Works within stack discipline

```c
jmp_buf env;
int error1 = 0;
int error2 = 1;
void foo(void), bar(void);

int main()
{
  switch(setjmp(buf)) {
  case 0:
    foo();
    break;
  case 1:
    printf("Detected an error1 condition in foo\n");
    break;
  case 2:
    printf("Detected an error2 condition in foo\n");
    break;
  default:
    printf("Unknown error condition in foo\n");
    exit(0);
  }
}
```

Limitations of Long Jumps (cont.)

- Works within stack discipline

```c
jmp_buf env;
p1()
{
  if (setjmp(env)) {
    /* Long Jump to here */
    p2();
  }
}
p2()
{
  if error1
    p1();
    p3();
}
p3()
{
  longjmp(env, 1);
}
```

### Example (cont)

```c
jmp_buf buf;
int error = 0;

void foo(void), bar(void);

int main()
{
  switch(setjmp(buf)) {
  case 0:
    foo();
    break;
  case 1:
    printf("Detected an error1 condition in foo\n");
    break;
  case 2:
    printf("Detected an error2 condition in foo\n");
    break;
  default:
    printf("Unknown error condition in foo\n");
    exit(0);
  }
}
```