Exceptional Control Flow

Hardware support for reacting to the rest of the world.
Control Flow

Processor: read instruction, execute it, go to next instruction, repeat

*Physical control flow*

```plaintext
<startup>
inst_1
inst_2
inst_3
...
inst_n
<shutdown>
```

Explicit changes:

Exceptional changes:
Exceptions

Synchronous: caused by instruction

*Traps: system calls*

*Faults: unintentional, maybe recoverable*

*Aborts: unintentional, unrecoverable*

Asynchronous (Interrupts): caused by external events
Exceptions: hardware support for OS

transfer control to OS in response to event

What code should the OS run?

![Diagram showing the flow of control from User Code to OS Kernel and back](image-url)
Interrupt Vector

in memory
special register holds base address

Exception Table

0
1
2
...
n-1

code for exception handler 0

code for exception handler 1

code for exception handler 2

code for exception handler n-1

... a jump table for exceptions...
Open a file (trap/system call)

```plaintext
0804d070 <__libc_open>:
  ...  
804d082: cd 80    int $0x80
804d084: 5b       pop %ebx
  ...  
```

User Code  OS Kernel

- `int`: Call system function
- `pop`: Save return address
- `exception`: Transfer control to kernel
- `returns`: Return to user program
- `open file`: Open file
Segmentation Fault

```c
int a[1000];
void bad () {
    a[5000] = 13;
}
```

Write to invalid memory location.

User Code

OS Kernel

movl

exception: page fault

detect invalid address

signal process

aborts process with SIGSEGV signal
Page Fault

Write to valid memory location
... but contents currently on disk instead
(more later: virtual memory)

```
int a[1000];
main () {
    a[500] = 13;
}
```

![Diagram of user code and OS kernel interaction]