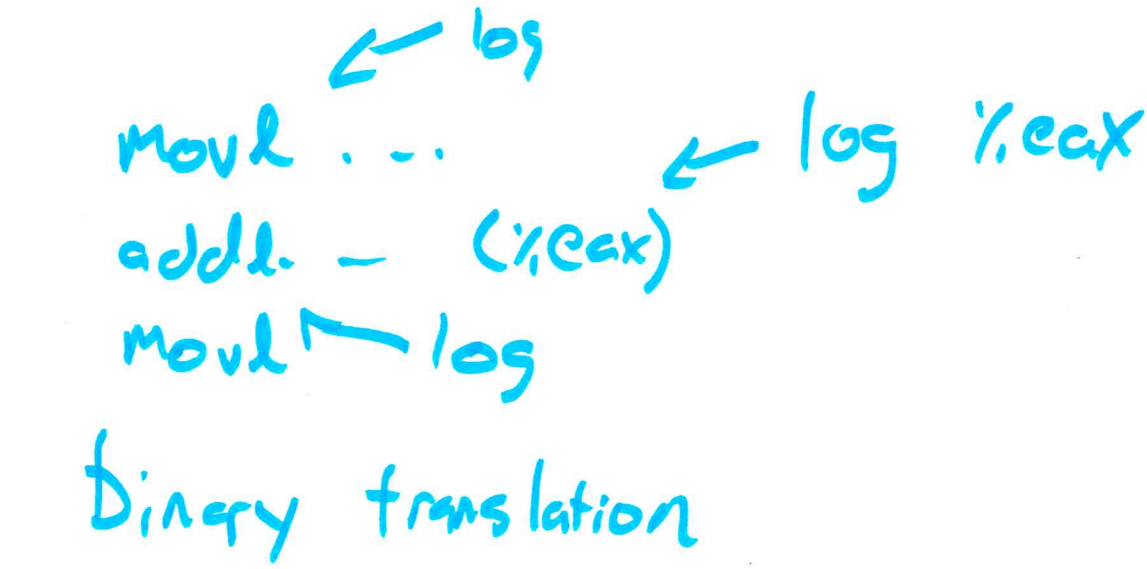
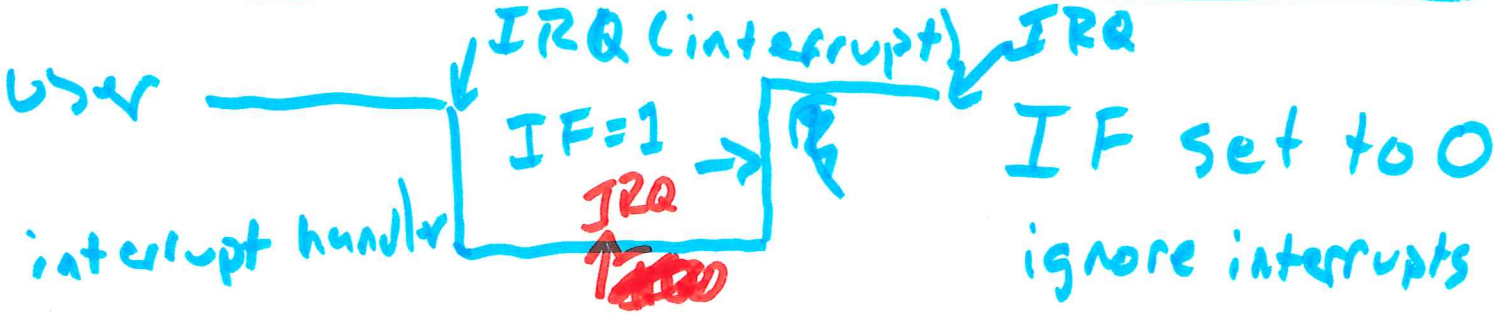


binary translation

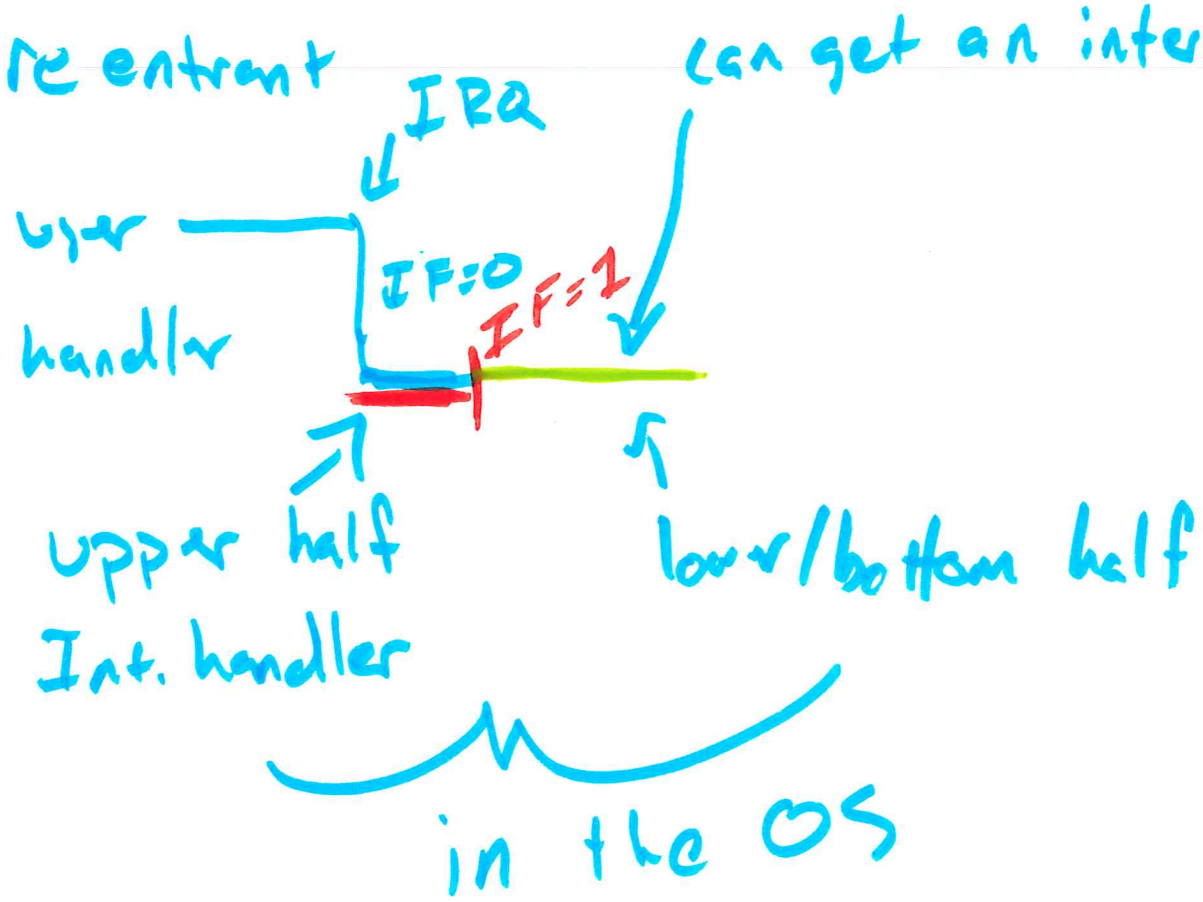
Exceptions

- interrupts → async
- traps → sync. (int) usr does on purpose
- faults → sync
  - ↳ page faults
  - ↳ divide by 0,
- abort → sync
  - fault that you can't fix

→ kill the process



↑ non-reentrant (IF=0 for whole interrupt handler)



~~Signal~~ Signals → "user-level interrupts/exceptions"

SIGINT (ctrl-c)
SIGKILL (kill -9)
SIGSEGV (segfault)

Number	Name	Default action	Corresponding event
1	SIGHUP	Terminate	Terminal line hangup
2	SIGINT	Terminate	Interrupt from keyboard
3	SIGQUIT	Terminate	Quit from keyboard
4	SIGILL	Terminate	Illegal instruction
5	SIGTRAP	Terminate and dump core (1)	Trace trap
6	SIGABRT	Terminate and dump core (1)	Abort signal from `abort` function
7	SIGBUS	Terminate	Bus error
8	SIGFPE	Terminate and dump core (1)	Floating point exception
9	SIGKILL	Terminate (2)	Kill program
10	SIGUSR1	Terminate	User-defined signal 1
11	SIGSEGV	Terminate and dump core (1)	Invalid memory reference (seg fault)
12	SIGUSR2	Terminate	User-defined signal 2
13	SIGPIPE	Terminate	Wrote to a pipe with no reader
14	SIGALRM	Terminate	Timer signal from `alarm` function
15	SIGTERM	Terminate	Software termination signal
16	SIGSTKFLT	Terminate	Stack fault on coprocessor
17	SIGCHLD	Ignore	A child process has stopped or terminated
18	SIGCONT	Ignore	Continue process if stopped
19	SIGSTOP	Stop until next SIGCONT (2)	Stop signal not from terminal
20	SIGTSTP	Stop until next SIGCONT	Stop signal from terminal
21	SIGTTIN	Stop until next SIGCONT	Background process read from terminal
22	SIGTTOU	Stop until next SIGCONT	Background process wrote to terminal
23	SIGURG	Ignore	Urgent condition on socket
24	SIGXCPU	Terminate	CPU time limit exceeded
25	SIGXFSZ	Terminate	File size limit exceeded
26	SIGVTALRM	Terminate	Virtual timer expired
27	SIGPROF	Terminate	Profiling timer expired
28	SIGWINCH	Ignore	Window size changed
29	SIGIO	Terminate	I/O now possible on a descriptor
30	SIGPWR	Terminate	Power failure

Figure 8.25 **Linux signals.** Notes: (1) Years ago, main memory was implemented with a technology known as *core memory*. "Dumping core" is a historical term that means writing an image of the code and data memory segments to disk. (2) This signal can neither be caught nor ignored.

A *signal* is a small message that notifies a process that an event of some type has occurred in the system. For example, Figure 8.25 shows the 30 different types of signals that are supported on Linux systems. Typing "`man 7 signal`" on the shell command line gives the list.

Each signal type corresponds to some kind of system event. Low-level hardware exceptions are processed by the kernel's exception handlers and would not

Signals are delivered by OS/kernel to your process

↳ program/app/code that's running

↳ a bunch of stuff in OS to make this possible

Signals are delivered asynchronously

user-level applications can define signal handlers → code that executes on a signal

programs/processes can send signals to one another

kill sends a signal to the PID specified

kill -9 1043z

↳ signal #9 to PID 1043z

to set a signal handler

Sigaction system call Man Sigaction

SIGALRM

SIGALRM

sigaction (int signum, → #of the signal

struct sigaction *act,

struct sigaction *~~oldact~~)

oldact

Struct sigaction {

Void (*sa_handler)(int);

pointer to function
that you want to call
when you recv. a signal

void func(int) {
}

Void (*sa_sigaction)(int, -, -)

sigset_t mask;

int sa_flags;