Interrupts

STOP!!! Do this!!!
Interrupts

- Have device tell OS it is ready

- OS can then interrupt what it is doing to:
  - Determine what device wants service
  - Determine what service it wants
  - Perform or start the service
  - Go back to what OS was doing
Examples of Interrupts

- Disk drive at sector/track position (old days)
- Mouse moved
- Keyboard key pressed
- Printer out of paper
- Video card wants memory access
- Modem sending or receiving
- USB scanner has data
Interrupts

Interrupt Properties

• They arrive asynchronously
• Can’t communicate with a running program (no args or return values)
• They are associated with various priorities
• You want to handle them soon (interrupt latency)
• (usually) want to resume program
Traps

Trap - Internal conditions related to instruction stream

Trap Examples:

- syscall
- Illegal instruction
- Arithmetic overflow
- “lw” from unaligned address
Traps

Trap Properties

• they arrive synchronously
• get trap in same place if you re-run program
• they are associated with various priorities
• must handle immediately
• (usually) want to resume program
Exceptions

Exception: mechanism used to handle both interrupts and traps
- HW handles initial reaction
- then invokes SW called an Exception Handler

HW (MIPS)
1. Sets state giving cause of exception
2. Changes to Kernel mode, saving the previous mode in a hardware stack – for dangerous work ahead.
3. Disables further interrupts – to prevent infinite loop
4. Saves current PC – to be able to resume previously running program
5. Jumps to hardware address 0x8000 0080
Exceptions

HW (HC11)

1. Wait for current instruction to complete
2. Disable further interrupts
3. Save all CPU registers and return address on stack
4. Access “interrupt vector table” according to source
5. Jump to where the interrupt routine is specified in table
6. Use RTI to return
MIPS Exceptions

MIPS Type Exception Handler

- Set flag to detect incorrect entry
- Saves some registers
- Ascertain exception type
  - exception_code in Cause register
- Jump to specific exception handler
- Handle specific exception
- To resume program, jump to clean up routine
- Restore registers
- Reset flag (that detects incorrect entry)
- Restore previous mode
- Re-enable interrupts
- Jump back to program (using Exception Program Counter (EPC))
MIPS Exceptions

**syscall handler**
- Change EPC to point to next instruction
  - resume after syscall
- Look at syscall number (MIPS: $2)
- Jump to specific syscall handler

**putc handler** (one of many syscall handlers)
- Add char (from $4) to putqueue
- Turn on display interrupts
- Try to print a char
- Jump back to main exception handler

**getc handler**
- Turn on keyboard interrupts
- Try to get char when interrupt comes
- Jump back to main exception handler with data in $2
MIPS Exceptions

External (I/O) interrupt handler

- Which interrupt
  - Interrupt field in Cause register
- Call specific interrupt handler
- Jump back to main exception handler
• **Kernel Registers k0 and k1**
  
  • GPR $26 ($k0) and $27 ($k1) (used to store EPC)
  • Kernel can hop in between any two user instructions

• **Coprocessor “C0” registers**

  • Status ($12), Cause ($13), EPC ($14)
  • mfc0 $s0, $14
    • reads C0 register $14 (EPC) and puts it in $s0
  • mtc0 $s0, $14
    • writes $s0 to C0 register $12 (Status)
**MIPS Exception Registers**

**Cause Register** – way hardware makes information available to software about pending interrupts.
- 6 hardware interrupt bits
- 2 software interrupt bits
- 4 bits of exception code

**Status Register** – contains information about the status of features of the computer that can be set by the processor when in kernel mode.
- Interrupt masks
- 3 mode/interrupt enable pairs
  - mode is user/kernel
  - on interrupts, stack mode/interrupt pairs
HC11 Exceptions

HC11 does things slightly different than on the MIPS.

Uses an interrupt vector table to find address of interrupt and goes straight to specific interrupt handler software.

Not as complex as MIPS.
HC11 Exceptions

;  1 unavailable to user  SCI
ISR_JUMP2 EQU $7BC5  ; SPI
ISR_JUMP3 EQU $7BC8  ; Pulse Accumulator Input
ISR_JUMP4 EQU $7BCB  ; Pulse Accumulator Overflow
;  5 unavailable to user  Timer Overflow
;  6 unavailable to user  Output Compare 5
ISR_JUMP7 EQU $7BD4  ; Output Compare 4
ISR_JUMP8 EQU $7BD7  ; Output Compare 3
ISR_JUMP9 EQU $7BDA  ; Output Compare 2
ISR_JUMP10 EQU $7BDE  ; Output Compare 1
ISR_JUMP11 EQU $7BE3  ; Input Capture 3
ISR_JUMP12 EQU $7BE6  ; Input Capture 2
ISR_JUMP13 EQU $7BE9  ; Input Capture 1
;  14 unavailable to user  Real Time Interrupt
ISR_JUMP15 EQU $7BEC  ; IRQ
;  16 unavailable to user  XIRQ
;  17 unavailable to user  SWI
;  18 unavailable to user  Illegal Opcode
ISR_JUMP19 EQU $7BF8  ; Cop fail
ISR_JUMP20 EQU $7BF9  ; Cop clock fail
;  21 unavailable to user  Reset (found at 0x8040)
**HC11 Exceptions**

* Some of the internal 64 registers are defined here. Their definitions should be self-evident from the Motorola manuals.

```plaintext
BAUD EQU REGBASE+$2B ; sci baud register
SCCR1 EQU REGBASE+$2C ; sci control1 register
SCCR2 EQU REGBASE+$2D ; sci control2 register
SCSR EQU REGBASE+$2E ; sci status register
SCDR EQU REGBASE+$2F ; sci data register
```

* Useful system variables
* These may change if the monitor software changed.
* O.K. ver2.17
* Addresses can be found in the linker map file.

```plaintext
_GETCHR EQU $8A0A ; char getchr (void)
IRQCOUNT EQU $0CD5 ; 16-bit integer that increments each time IRQ is called
_PUTCHR EQU $89E5 ; char putchar (char c)
TICSEC EQU $02E3 ; 16-bit integer that increments once each second
```
HC11 Exceptions

_WAIT EQU $8A37 ; software timer 4.096 ms per tick, argument in x-reg

USERTICK EQU $02D7 ; unsigned int incremented every 4.096ms by system clock
Multiple Exceptions

Problem: How about multiple simultaneous exceptions?

Solution: Have priorities in HW / SW

• Handle highest-priority exception first
• Equal priority exceptions handled arbitrarily
• Give higher priority
  • more serious (e.g., power failing)
  • can’t wait long (e.g., rotating disk)
Multiple Exceptions

How about exceptions during exception handling?

- Make it wait until done with first exception
  - May be a bad idea for higher priority exception

- Make exception handler re-entrant
  - Make able to handle multiple active calls
  - Allow higher-priority exceptions to bypass lower-priority exception currently being serviced
Implementing a Re-entrant exception handler

- Initial exception disables all interrupts
- Exception handler (EH) determines exception’s priority
- EH saves any state that could be clobbered by higher priority interrupts (e.g. EPC)
- EH re-enables higher-priority interrupts
- Higher-priority interrupts may or may nor occur
- This EH eventually finishes
Multiprogramming – the ability to run one or more programs “simultaneously” on one CPU.

- An active program is called a *process* or *task*.
- A processes’ state is:
  - program counter
  - registers
  - memory locations being used
  - Some OS memory
    - EPC
    - other registers
Multiprogramming

- OS has one exception handler called the “kernel”
- On an exception, CPU jumps into the kernel
- Kernel will eventually resume the user process

- If the user process needs I/O (disk read)
  - Kernel schedules it
  - 20ms is ~ 2 million instructions
  - Start (or switch to) another task (multitasking)

- If user process does not need I/O
  - Kick it out (context switch) after some amount of time
  - Every X clock interrupts and switch again
OS has several job queues - An entry is the complete state of a process

Some queue examples:

- A queue of ready jobs with priority
- A queue of I/O device(s)
  - Jobs are moved to ready queue when I/O complete
- A queue of sleeping or halted jobs

OS picks jobs from the ready queue as appropriate
Generalized Exceptions

1. User program running
2. Exception is generated
   • Internal/External
   • Trap/Interrupt
3. Determine source of exception
   • Requires a bus cycle for some peripheral bus architectures.
4. Save return PC and any status registers modified by hardware
   • MIPS: EPC, Control, Status
   • HC11: program counter, index registers, acc.
5. Jump to exception handler
   • MIPS: Single EH uses interrupt and exceptions type bits to find correct procedure
   • HC11: Jump table starting at 0xFFD6
   • 68000: Jump to location specified by interrupt device – *vectored interrupts*

6. Save state used by exception handler

7. Go to specific case
   • Check exception type
   • Query device
   • Check syscall register
Generalized Exceptions

8. Process Exception
   • Enable higher-priority interrupts if possible
   • For interrupt, clear device’s interrupt flag
   • Check device for multiple conditions

9. Restore state used by exception handler

10. Return from exception
    • Restore mode
    • Restore hardware-saved state
    • Jump back to user program
    • or exception handler
Exceptions (interrupts and traps) are a combination of hardware and software.

The hardware detects the exception and then calls software to handle it.

Much more efficient than polling but requires specialized hardware.