Midterm Thursday in class OBONNS.

on HW & Labs

Next checkpoint: 7 weeks

Interrupts

Exception Interrupt: 

EPOR PC: 
PC = 0x8000 0180

- finish current instruction
- save PC
- load address of int handle into PC
- procedure if no args, no return
- set some bits to indicate what caused INT

return put old PC back

Laid-off 1008 
JAL plus
100C
addi $50, $V0, 0

plus: 2000
add $V0, $a1, $a2
jr $ra
nop

We said that the top was OK (swapped from jnl plus
nop
addi $50)

why is that wrong? $V0 not ready yet

when used in 100C

on MIPS

coprocessor 0 (separate registereds)

CP0 reg status

IM-In, Mask
IE-In

CP0 reg cause

IP-Exception PC

$9
Count
$11
Compare
Exceptions (Interrupts, traps, ...)

- Keyboard
- Mouse/touchpad
- Timer/RTC
- Buttons
- ALU overflow, ÷ 0
- Radio(s)
- Ethernet
- SPI, I2C, ...
- Timers

why?
why? asynchronous/unpredictable event

blocking: good for ASCII text input.
not so good for video games.
polling: OK if you have a tight loop.
very difficult to guarantee service otherwise.
soln: have hardware cause a procedure call any time it wants.

issues, problems:
- procedure call can't use $ra during call
  or any other register w/o saving.
  (including $t0-$9, $at, ...)
- need a way for procedure to know
  what happened.
  on MIPS: cause, status, EPC registers

like scanf
while (numinp) wait;
process_input()
new instruction
mfc #p move from mt.c #p move to rt is mips register
rd is c#p register
opcode = 16
rs = 0 from rs = 4 to
when $IE & (IM & IP) then interrupt
when $IE = 0

Causes
cause (5) timer
set when $count = compare

cause (14) RTC
set when counter rolls over to $p

cause (10) UART
set whenever UART TX is done

cause (6:2)

Ex: 80000180:

mfc #p, $ko, cause
and $ko, $ko, ox0000f

mfc #p, $k1, status
and $ko, $ko, $k1
// now we have unmasked cause pending interrupts

check each bit and j (chandb)
store bit of handler

andi $k1, $ko, 1 << 15
beg $k1, $ko, timer - IV
andi $k1, $ko, 1 << 14
beg $k1, $ko, RTC - IV

interrupt finish "current" inst
EPC <- PC
PC <- 0x8000 0180 (Int Vector, IV)
Status(0) = 0
IE

Code @ IV is int. handler
- figure out which interrupt(s)
- clear w/ it
- reset bit(s) of IP
- ret

IE = 0
PC <- EPC

// ISR_RTC

// happens every 2^32 cycles = 1/min?

RTC_INT:

nop
add $k1, $k1, 1
sw $k1, shl_time
mfc $sp, $k1, Cause
and $k1, $k1, 1 << 14

mfc $sp, $k1, Cause

en thrilling

assemble
our compiler may not do this.

old
mfc $sp, $k0, EPC
jr $k0
new
mfc $sp, $k0, status

temp $k0, $k0, 1
mfc $sp, $k0, status

interrupt handler
no args, no return
must save any registers they touch (even $t)
keep short!
this is why you pre-decrement

SP

Lets you write code oblivious to device I/O

while (!done)

calc();
draw();

ISR_timer() ISR_UART() ISR_RTC()

---

System calls often use SW ind (except trap)
- return control to supervisor/kernel
- check permissions, etc.
- in this case arguments are placed in known locations, e.g. A0...
and you can have a return

Thread/process control:
give control to kernel every 10ms.

thread/process control:
give control to kernel every 10ms.

high level example

eventolare (ev)

switch (ev.type) {

case mouseMove:

case KeyPress: