GA13.4. Add a keyboard interrupt

The TAs will review your code and award credit based on thoughtful attempts to solve the problem.

Suppose we want to add a keyboard interrupt to the interrupt handler. The keyboard provides a value indicating what key was pressed at memory-mapped I/O address `0xFFFF0000`. The enable/status bit for the keyboard interrupt can be found at `0x4000`.

Add some interrupt handling that

1. copies the value of the key pressed from the MMIO address and writes that value to the address pointed at by the user's global pointer (`$gp`)
2. Set a flag for the user that tells them that a key was pressed
3. acknowledge the keyboard interrupt by writing to address `0xFFFF000c`

Make sure that you respect the calling conventions of the interrupt handler. You do not need to write any code in the user program since we are not concerned with what the user will do with the keyboard input in this question.

interrupt_handler.s

```
.data  # Space for global variables that users and kernel can access
##
has_bonked: .byte 0                # bonk flag
has_timer:  .byte 0                # timer flag

.text
main:                              # ENABLE INTERRUPTS
  li     $t4, 0x8000               # timer interrupt enable bit
  or     $t4, $t4, 0x1000          # bonk interrupt bit
  or     $t4, $t4, 1               # global interrupt enable
  mtc0   $t4, $12                  # set interrupt mask (Status register)

                                   # REQUEST TIMER INTERRUPT
  lw     $v0, 0xffff001c($0)       # read current time
  add    $v0, $v0, 50              # add 50 to current time
  sw     $v0, 0xffff001c($0)       # request timer interrupt in 50 cycles

  li     $a0, 10                   # set spimbot velocity to 10
  sw     $a0, 0xffff0010($zero)    # drive

infinite:


  # Respond to a bonk
  la      $t5, has_bonked          # get location of flag for bonks
  lb      $t6, 0($t5)              # check current value of bonk
  beq     $t6, $zero, no_bonk      # check if bonk flag is 0 or 1
  li      $a0, 180                 # ???
  sw      $a0, 0xffff0014 ($zero)  # ???
  li      $a1, 0                   # ???
  sw      $a1, 0xffff0018 ($zero)  # ???

  sb      $zero, 0($t5)            # reset bonk flag
no_bonk:

  # Respond to a timer
  la      $t5, has_timer           # get location of flag for timer
  lb      $t6, 0($t5)              # check current value of timer
  beq     $t6, $zero, no_timer     # check if timer flag is 0 or 1
  li      $a0, 90                  # ???
  sw      $a0, 0xffff0014($zero)   # ???
  sw      $zero, 0xffff0018($zero) # ???

  lw      $a1, 0xffff001c($0)      # current time
  add     $a1, $a1, 50000
  sw      $a1, 0xffff001c($0)      # request timer in 50000

  sb      $zero, 0($t5)            # reset timer flag
no_timer:

 j infinite

.kdata                # interrupt handler data (separated just for readability)
                      # space for global variables that only the kernel can access
chunkIH:.space 8      # space for two registers
                      # we will use this space as a pseudo-stack
non_intrpt_str:   .asciiz "Non-interrupt exception\n"
unhandled_str:    .asciiz "Unhandled interrupt type\n"

.ktext 0x80000180
interrupt_handler:
.set noat
  move    $k1, $at                 # Save $at
.set at
  la      $k0, chunkIH             # Set $k0 to point to chunkIH in kernel data segment
                                   # we will treat $k0 as a pseudo-stack-pointer
  sw      $a0, 0($k0)              # Get some free registers
  sw      $a1, 4($k0)              # by storing them to a global variable

  mfc0    $k0, $13                 # Get Cause register
  srl     $a0, $k0, 2              # Shift so ExcCode is least-significant bits
  and     $a0, $a0, 0x1f           # Mask to reveal ExcCode field
  bne     $a0, 0, non_intrpt       # Branch on whether there is an exception

interrupt_dispatch:                # Interrupt:
  mfc0    $k0, $13                 # Get Cause register, again
  beq     $k0, $zero, done         # handled all outstanding interrupts

  and     $a0, $k0, 0x1000         # is there a bonk interrupt?
  bne     $a0, 0, bonk_interrupt

  and     $a0, $k0, 0x8000         # is there a timer interrupt?
  bne     $a0, 0, timer_interrupt

  # add dispatch for other interrupt types here.
  li      $v0, 4                   # Unhandled interrupt types
  la      $a0, unhandled_str
  syscall
  j       done

bonk_interrupt:                    # spimbot bonked into a wall
  sw      $a1, 0xffff0060($zero)   # acknowledge interrupt

  la      $t0, has_bonked          # get address of bonk flag
  li      $t1, 1                   # prep 1 to change bonk flag
  sb      $t1, 0($t0)              # write 1 to tell user about bonk

  j       interrupt_dispatch       # see if other interrupts are waiting

timer_interrupt:                   # spimbot's timer alarm went off
  sw      $a1, 0xffff006c($zero)   # acknowledge interrupt

  la      $t0, has_timer           # get address of timer flag
  li      $t1, 1                   # prep 1 to change timer flag
  sb      $t1, 0($t0)              # write 1 to tell user about timer

  j       interrupt_dispatch       # see if other interrupts are waiting

non_intrpt:                        # an exception occurred (Oh No!)
  li      $v0, 4
  la      $a0, non_intrpt_str
  syscall                          # print out an error message
  j       done

done:
  la      $k0, chunkIH
  lw      $a0, 0($k0)              # Restore saved registers
  lw      $a1, 4($k0)
.set noat
  move    $at, $k1                 # Restore $at
.set at
  eret                             # Return from exception handler
```

Restore original file

Save

GA 13

Assessment overview

Total points: 80/100

Score: 80%

Question GA13.4

Total points: — /5

Manually-graded question

Report an error in this question

Previous question Next question

Personal Notes

No attached notes

Attach a file

Add text note