Writing an Embedded Controller

Embedded Controller

- A ‘computer’ with a dedicated function within a larger system
- Usually has ‘real time computing’ constraints
  - Is expected to process requests at a certain speed
  - Example, TV controllers are expected to respond to and typically fully process each user input within at most a few hundred milliseconds
- Examples:
  - Digital watches, MP3 players, traffic lights, cars, TVs, Washing machines, game consoles, printers, routers, etc.

Writing an Embedded Controller

- It usually consists of two parts
  - Initialization
  - A main loop
- More challenging than HW3 because HW3 is stateless, the microcontroller has **states**.
  - A state based program has to keep track of various global variables and program execution can produce different results each time it is run
  - Stateless programs produce the same output during each invocation and at expected to not have any global variables
- Experience:
  - The main loop usually has to deal with many things. It is important NOT to stay in any job for too long. You should process an event and almost immediately return to the main loop.

A Simple Program to Get Started

- Write a program which
  - Prints out “TV is working” every 3 seconds
  - Print out the ASCII of any key you have pressed immediately.

Two Jobs

- The simple program has two jobs:
  1. A periodical job executed every 3 seconds
  2. A job to process the input
- Note:
  - Cannot sleep for 3 seconds and then print out the sentence because cannot process the input while sleeping
  - Must make sure that each iteration of the main loop is short, such that you can check at a fine time granularity if
    - need to print status
    - Has new keyboard input

The code should look like

```
loop:  if key pressed
           print ascii value
       if 3 sec timer expires
           print msg
       goto loop
```

```
        .kdata                                  # kernel data
s1:     .word 10                                # storage for register $v0
s2:     .word 11                                # storage for register $a0
new_line:
        .asciiz "\n"
msg_tvworking:
        .asciiz "TV is working\n"

.ktext 0x80000180                               # kernel interupt code starts here

        sw $v0, s1                              # Save registers
        sw $a0, s2

        mfc0 $k0, $13                           # Cause register
        srl $a0, $k0, 2                         # Extract ExcCode Field
        andi $a0, $a0, 0x1f

        bne $a0, $zero, kdone                   # Exception Code 0 is I/O. Only processing I/O here

        lui $v0, 0xFFFF                         # $t0 = 0xFFFF0000;
        lw $s6, 4($v0)                          # get the input key, using $s6 as destination

kdone:
        lw $v0, s1                              # Restore registers
        lw $a0, s2

        mtc0 $0, $13                            # Clear Cause register
        mfc0 $k0, $12                           # Set Status register
        andi $k0, 0xfffd                        # clear EXL bit
        ori  $k0, 0x11                          # Interrupts enabled
        mtc0 $k0, $12                           # write back to status

        eret                                    # return to EPC
        .text
        .globl main
main:   mfc0 $a0, $12               # read from the status register
        ori $a0, 0xff11             # enable all interrupts
        mtc0 $a0, $12               # write back to the status register

        lui $t0, 0xFFFF             # $t0 = 0xFFFF0000;
        ori $a0, $0, 2              # enable keyboard interrupt
        sw $a0, 0($t0)              # write back to 0xFFFF0000;

        li $s0, 300                 # 3 sec counter
        li $s6, 10000               # $s6 used to pass the ascii code
        li $s7, 10000               # a number that can't be in ascii code

loop:   beq $s6, $s7, mainloopnext1
        ori $a0, $s6, 0
        li $s6, 10000               # reset $s6

        li $v0,1                    # print it here.
        syscall

        li $v0,4                    # print the new line
        la $a0, new_line
        syscall
```

```
mainloopnext1:
        addi $s0, $s0, -1
        bne $s0, $0, mainloopnext4
        li $s0, 300

        la $a0, msg_tvworking
        li $v0, 4
        syscall

mainloopnext4:
        jal delay_10ms
        j loop

        li $v0, 10                  # exit,if it ever comes here
        syscall

delay_10ms:
        li $t0, 6000                # arbitrary value, attempts to busy
delay_10ms_loop:                    # loop for 10ms; may need
        addi $t0, $t0, -1           # to change $t0 for your computer
        bne $t0, $0, delay_10ms_loop
        jr $ra
```

A Slightly More Advanced Version

- Write a process_input function that responds to `m’, `h’, `q’ (ascii code 109, 104, 112, respectively).
- Basically, The TV is initially not in the ``menu state.’’ When the user presses `m’ while the TV is not in the menu state, the TV should show a very simple menu, and enters the menu state:
  - “`h' to print hello, `q' to quit.”
- In the menu state,
  - if the user presses `h’, print out “Hello!”
  - if the user presses `q’, print out “quit” and quits the menu state.
- If not in the menu state,
  - the TV does not respond to `h’ and `q’.

The Challenge

- How do you know whether to respond to ‘h’ or ‘q’ or not?
  - Should not respond in the normal state
  - Should respond under menu
- A naïve way is to write a process_input function that
  - Called when ‘m’ is pressed then waits there for ‘h’ and ‘q’
  - Problem?

The solution

- Maintain a global variable to remember if we are in the menu state
- Write the process_input function by checking the variable first

```
        .kdata
s1:     .word 10
s2:     .word 11

        .data
menuLevel:
        .word 0

msg_tvworking:
        .asciiz "tv is working\n"

msg_menu:
        .asciiz "`h' to print hello, `q' to quit.\n"

msg_hello:
        .asciiz "hello!\n"

msg_quit:
        .asciiz "quit.\n“
        .ktext 0x80000180                  # kernel code starts here

        sw $v0, s1                         # Save registers

        mfc0 $k0, $13                      # Cause register
        srl $a0, $k0, 2                    # Extract ExcCode Field
        andi $a0, $a0, 0x1f

        bne $a0, $zero, kdone              # Exception Code 0 is I/O. Only processing I/O here

        lui $v0, 0xFFFF                    # $t0 = 0xFFFF0000;
        lw $s6, 4($v0)                     # get the input key

kdone:
        lw $v0, s1                         # Restore registers
        lw $a0, s2

        mtc0 $0, $13                       # Clear Cause register
        mfc0 $k0, $12                      # Set Status register
        andi $k0, 0xfffd                   # clear EXL bit
        ori  $k0, 0x11                     # Interrupts enabled
        mtc0 $k0, $12                      # write back to status

        eret                               # return to EPC
        .text
        .globl main
main:   mfc0 $a0, $12                   # read from the status register
        ori $a0, 0xff11                 # enable all interrupts
        mtc0 $a0, $12                   # write back to the status register

        lui $t0, 0xFFFF                 # $t0 = 0xFFFF0000;
        ori $a0, $0, 2                  # enable keyboard interrupt
        sw $a0, 0($t0)                  # write back to 0xFFFF0000

        li $s0, 300                     # 3 secs
        li $s6, 10000                   # $s6 used to pass the ascii code
        li $s7, 10000                   # a large number impossible to be an ascii code

mainloop:
        # 1. read keyboard input, and process it
        beq $s6, $s7, mainloopnext1
        ori $a0, $s6, 0
        li $s6, 10000                   # $s0 used to pass the ascii code
        jal process_input

mainloopnext1:
        addi $s0, $s0, -1
        bne $s0, $0, mainloopnext2
        li $v0, 4
        la $a0, msg_tvworking
        syscall
        addi $s0, $0, 300
mainloopnext2:
        jal delay_10ms
        j mainloop

        li $v0,10 # exit
        syscall
```

```
delay_10ms:
        li $t0, 6000
delay_10ms_loop:
        addi $t0, $t0, -1
        beq $t0, $0, delay_10ms_done
        j delay_10ms_loop
delay_10ms_done:
        jr $ra
```

```
process_input:
        la $t0, menuLevel
        lw $t1, 0($t0)
        bne $t1, $0, pi_menu_L_1

        li $t0, 109                                     # comparing with the ascii of `m'
        bne $a0, $t0, process_input_done
        la $t0, menuLevel
        li $t1, 1
        sw $t1, 0($t0)
        la $a0, msg_menu
        li $v0, 4
        syscall
        j process_input_done

pi_menu_L_1:
        li $t0, 104                                     # comparing with the ascii of `h'
        bne $a0, $t0, pi_menu_L_1_comp_q
        la $a0, msg_hello
        li $v0, 4
        syscall
        j process_input_done
pi_menu_L_1_comp_q:
        li $t0, 113                                     # comparing with the ascii of `q'
        bne $a0, $t0, process_input_done
        la $a0, msg_quit
        li $v0, 4
        syscall
        la $t0, menuLevel
        sw $0, 0($t0)
        j process_input_done

process_input_done:
        jr $ra
```