DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2026)

Ayudantía 9: I/O + RISC-V

Ayudantes: Ignacio Gajardo (gajardo.ignacio@uc.cl), Mario Rojas (mario.denzel@estudiante.uc.cl), Nicolás Romo Aubele (nroma@uc.cl), José Mendoza (jfmendoza@uc.cl)

Pregunta 1: Preguntas Conceptuales

(a) Explique el uso e importancia de la señal `IRQ` enviada por el controlador de interrupciones a la `CPU`.

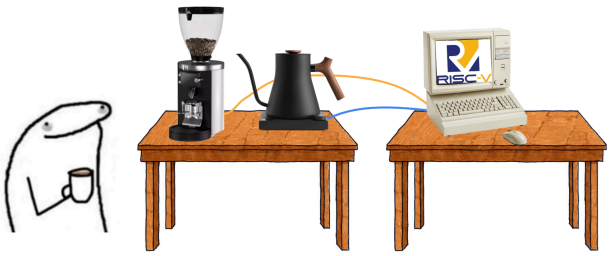
(b) ¿Por qué una *Interrupt Service Routine* (ISR) debe respaldar y restaurar el estado de la CPU al iniciar y antes de finalizar su ejecución, respectivamente?

(c) ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido?

(d) A diferencia del esquema *memory-mapped*, ¿por qué un dispositivo I/O con comunicación *port-mapped* necesita modificar la ISA con instrucciones adicionales?

Pregunta 2: Comunicación con dispositivos I/O

Suponga que decide, con el propósito de ahorrar tiempo, automatizar la preparación de café filtrado a partir de un molinillo y hervidor eléctricos conectados a un computador con ISA RISC-V. A continuación, una representación gráfica de su implementación:



Para la comunicación con los electrodomésticos, sus registros de 32 bits son mapeados en memoria desde la dirección 0x600DCAFE:

Offset	Nombre	Descripción
0x00	`coffee_grinder_status`	Registro estado molinillo
0x04	`coffee_grinder_command`	Registro comando molinillo
0x08	`coffee_grinder_grind_size`	Config. grosor molienda
0x0C	`coffee_grinder_weight`	Config. gramos a moler
0x10	`kettle_status`	Registro estado hervidor
0x14	`kettle_command`	Registro comando hervidor
0x18	`kettle_temperature`	Config. temperatura hervidor (ºC)

Nombre	Descripción	Valor
`coffee_grinder_status`	Apagado	0
`coffee_grinder_status`	Encendido	1
`coffee_grinder_status`	Moliendo	2
`kettle_status`	Apagado	0
`kettle_status`	Encendido	1
`kettle_status`	Hirviendo	2
`coffee_grinder_command`	Encender	1
`coffee_grinder_command`	Apagar	255
`coffee_grinder_command`	Moler	128
`kettle_command`	Encender	1
`kettle_command`	Apagar	255
`kettle_command`	Hervir	128
`coffee_grinder_grind_size`	Fino	1
`coffee_grinder_grind_size`	Medio	2
`coffee_grinder_grind_size`	Grueso	3

Pensando en su preparación favorita, desarrolla el siguiente programa:

```
coffee_system:
# Carga 0 + 0x600DCAFE en t0
addi t0, zero, 0x600DCAFE
addi t2, zero, 0
addi t3, zero, 0
addi s1, zero, 1
addi s2, zero, 2
addi s3, zero, 128
addi s4, zero, 255
addi s5, zero, 30
addi s6, zero, 92

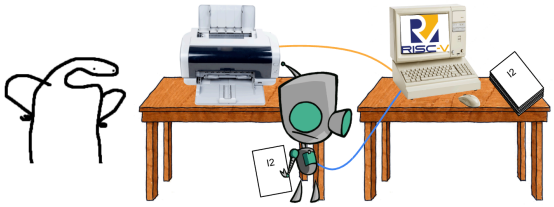
step_one:
    lw t1, 0(t0) # Carga Mem[t0 + 0] en t1
    # Salta a continue_step_one si t1 != s2
    bne t1, s2, continue_step_one
    jal ra, step_five # Llamado a step_five
    beq zero, zero, step_three
    continue_step_one:
        beq t1, s1, step_two
        sw s1, 4(t0) # Carga s1 en Mem[t0 + 4]
step_two:
    sw s2, 8(t0)
    sw s5, 12(t0)
    sw s3, 4(t0)
    jal ra, step_five
# Continúa en step_three -->
```

```
step_three:
    lw t1, 16(t0) # Carga Mem[t0 + 16] en t1
    # Salta a continue_step_three si t1 != s2
    bne t1, s2, continue_step_three
    jal ra, step_six
    beq zero, zero, end # Salto incondicional
    continue_step_three:
        beq t1, s1, step_four
        sw s1, 20(t0)
step_four:
    sw s6, 24(t0)
    sw s3, 20(t0)
    jal ra, step_six
    beq zero, zero, end
step_five:
    while_step_five:
        lw t1, 0(t0)
        beq t1, s2, while_step_five
    sw s4, 4(t0)
    jalr zero, 0(ra) # Retorno
step_six:
    while_step_six:
        lw t1, 16(t0)
        beq t1, s2, while_step_six
    sw s4, 20(t0)
    jalr zero, 0(ra)
end: # Término del programa.
```

Describa, basándose en las tablas provistas, las tareas que realiza el programa en cada *label* `step_*`. Haga una descripción según los estados y comandos; no una explicación por cada instrucción del programa.

Pregunta 3: Comunicación con dispositivos I/O II

Suponga que es el/la docente del curso y debe imprimir la I2. No obstante, la redacta a última hora y no alcanza a usar los equipos del DCC, por lo que decide imprimir en su hogar con un sistema innovador: una impresora y un robot conectados a un computador con ISA RISC-V. A continuación, una representación gráfica de su implementación:



Para llevar a cabo la comunicación, los registros de 32 bits de los dispositivos se mapean en memoria desde la dirección 0x00FF10AD:

Offset	Nombre	Descripción
0x00	`printer_status`	Registro de estado impresora
0x04	`printer_command`	Registro de comando impresora
0x08	`printer_copies`	Configuración de cantidad de copias impresora
0x0C	`printer_color`	Configuración de color impresora
0x10	`robot_status`	Registro de estado robot
0x14	`robot_command`	Registro de comando robot

Nombre	Descripción	Valor
`printer_status`	Apagado	0
`printer_status`	Encendido	1
`printer_status`	Imprimiendo	2
`robot_status`	Apagado	0
`robot_status`	Encendido	1
`robot_status`	Trabajando	2
`printer_command`	Encender	1
`printer_command`	Apagar	255
`printer_command`	Imprimir	128
`printer_color`	Blanco y negro	1
`printer_color`	Color	2
`robot_command`	Encender	1
`robot_command`	Rellenar papeles impresora	2
`robot_command`	Rellenar tinta impresora	3
`robot_command`	Apilar impresiones*	4
`robot_command`	Apagar	255

* Esta acción hace que el robot pase a estado “Trabajando”, el resto son instantáneas y no cambian su estado.

Para automatizar la impresión, desarrolla el siguiente programa:

```
printing_system:
# Carga 0 + 0x00FF10AD en t0
addi t0, zero, 0x00FF10AD
addi s1, zero, 1
addi s2, zero, 2
addi s3, zero, 3
addi s4, zero, 4
addi s5, zero, 128
addi s6, zero, 255
addi s7, zero, 60

step_one:
    lw t1, 16(t0) # Carga Mem[t0 + 16] en t1
    # Salta a continue_step_one si t1 != zero
    bne t1, zero, continue_step_one
    sw s1, 20(t0) # Carga s1 en Mem[t0 + 20]
    continue_step_one:
        lw t1, 16(t0)
        # Salta a continue_step_one si t1 == s2
        beq t1, s2, continue_step_one

step_two:
    lw t1, 0(t0)
    bne t1, zero, continue_step_two
    sw s1, 4(t0) # ---> Continúa a la derecha
```

```
# --> Continúa aquí
    continue_step_two:
        lw t1, 0(t0)
        beq t1, s2, continue_step_two

step_three:
    sw s7, 8(t0)
    sw s3, 20(t0)
    sw s1, 12(t0)

step_four:
    sw s5, 4(t0)
    continue_step_four:
        lw t1, 0(t0)
        beq t1, s2, continue_step_four

step_five:
    sw s4, 20(t0)
    continue_step_five:
        lw t1, 16(t0)
        beq t1, s2, continue_step_five

step_six:
    sw s6, 4(t0)
    sw s6, 20(t0) # Término del programa.
```

Describa, basándose en las tablas provistas, las tareas que realiza el programa en cada *label* `step_*`. Haga una descripción según los estados y comandos; no una explicación por cada instrucción del programa.

Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.

