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

IIC2343 - Arquitectura de Computadores (II/2025)

Guía de ejercicios: I/O

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), José Mendoza (jfmendoza@uc.cl)

Pregunta 1: Preguntas conceptuales

(a) Indique qué tipo de *mapping* debe tener un dispositivo I/O de almacenamiento dentro de un computador para que pueda interactuar con el controlador de DMA.

> **Solución:** Los valores de los registros de origen y destino del controlador de DMA deben ser direcciones del bus compartido para poder realizar la transferencia de datos entre la memoria y dispositivos I/O de almacenamiento. Por esto, el controlador solo puede acceder a direcciones de dispositivos *memory mapped*

(b) Si un computador ya tiene dispositivos *memory mapped* a través de un *address decoder*, ¿se necesita *hardware* adicional para habilitar la comunicación via *polling*? Justifique.

> **Solución: No**, no es necesario ya que en la comunicación via *polling* es la `CPU` la que se encarga de interactuar con los dispositivos a través de la revisión de sus estados, lo que ya puede realizar con el acceso directo a las direcciones de memoria asociadas a ellos.

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

> **Solución:** La función del vector de interrupciones es permitir la ejecución correcta de la `ISR` (*Interruption Service Routine*) a partir del identificador del dispositivo a ser atendido, lo que es gestionado por el controlador de interrupciones. Este vector contiene los **punteros** de las `ISR`, por lo que si se desea atender al dispositivo `i`, se ejecuta el llamado de la subrutina a partir de su dirección igual a `vector_interrupciones[i]`.

(d) Explique el significado y uso del identificador que envía el controlador de interrupciones luego de recibir la señal `INTA` por parte de la `CPU`.

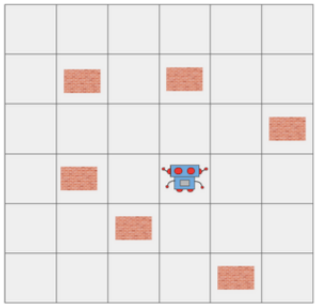
Solución: Al recibir la señal `INTA`, la `CPU` espera obtener el identificador asociado al **dispositivo cuya interrupción será atendida.** Este será enviado por el controlador de interrupciones. Luego, la `CPU` hace uso de este valor como índice del vector de interrupciones para obtener el puntero de la `ISR` del dispositivo atendido, permitiendo así la ejecución de su subrutina.

(e) Cuando usted trabaja en su computador, puede seguir haciendo uso del mismo mientras transfiere documentos de este a dispositivos externos y vice-versa. ¿A qué se puede deber esto? Justifique su respuesta en base a argumentos relativos a la comunicación entre la CPU y dispositivos I/O.

Solución: Esto se debe al uso del controlador DMA (*Direct Memory Access*). Este se encarga de llevar acabo la transferencia de datos entre los dispositivos I/O y la memoria del computador, evitando que la CPU ocupe tiempo de cómputo en esto y, permitiendo así, la ejecución de otros programas.

Pregunta 2: Comunicación con dispositivos I/O

(a) Un robot simple, conectado a un computador, es accesible mediante memory mapping. Este robot se mueve en un espacio cuadrado infinito, en el cual cada grilla puede estar vacía o contener una muralla. Este robot posee registros internos mapeados en memoria desde la dirección base 0x10101000. A continuación, se muestra una ilustración de este ejemplo y las tablas de direcciones y valores de estado y comando del robot:



Offset	Nombre	Descripción
0x00	`robot_stat`	Registro estado
0x04	`robot_comm`	Registro comando
0x08	`robot_turns`	Contador giros

Nombre	Descripción	Valor
`robot_stat`	Recién encendido	0
`robot_stat`	Nada que informar	255
`robot_stat`	Espacio libre adelante	1
`robot_stat`	Muralla adelante	2

Nombre	Descripción	Valor
`robot_comm`	Encender	255
`robot_comm`	Apagar	0
`robot_comm`	Avanzar	1
`robot_comm`	Giro antihor. en 90º	2
`robot_comm`	Examinar	4

En resumen, el robot tiene comandos para ser prendido, apagado, avanzar un espacio hacia adelante, girar en sentido antihorario en 90º o examinar lo que tiene delante de él. Cada vez que el robot se encuentra desocupado, i.e. que ha sido recién iniciado o ha terminado una acción, genera una interrupción para informar que es posible darle un nuevo comando. Adicionalmente, cuenta con un registro **robot_turns** que indica la cantidad de veces que ha girado. Este se utiliza para evitar que el robot retroceda, i.e. **para evitar que avance si ha girado dos veces**. A continuación, se muestra el fragmento de la ISR en RISC-V que plasma el control del robot:

```
ISR_robot:
        li t0, 0x10101000  # Carga en el registro t0 el valor del literal 0x10101000
    step_one:
        li s0, 0
        li s1, 255
        li s2, 1
        lw t1, 0(t0)
        beq t1, s0, step_two
        beq t1, s1, step_three
        beq t1, s2, step_four
        j step_six    # Salto incondicional a step_six
    step_two:
        li t2, 0
        sw t2, 8(t0)
    step_three:
        li t2, 4
        sw t2, 4(t0)
        j end_isr
    step_four:
        li t2, 2
        lw t3, 8(t0)
        beq t2, t3, step_six
    step_five:
        li t2, 0
        sw t2, 8(t0)
        li t2, 1
        sw t2, 4(t0)
        j end_isr
    step_six:
```

```
        lw t2, 8(t0)
        addi t2, t2, 1
        sw t2,8(t0)
        li t2, 2
        sw t2,4(t0
    end_isr:
        mret  # Retorno ISR.
```

Describa, basándose en las tablas provistas, las tareas que realiza el robot en cada *label* `step_*` dentro de la **ISR**. Considere que la instrucción **mret** es equivalente al retorno, pero utilizado en el contexto de interrupciones. Puede obviar el respaldo de los registros.

Solución:

A continuación, se describe lo que hace el robot en cada **step**:

1. **step_one**: El robot revisa su estado y, en base a eso, evalúa qué acción tomar: Si está recién encendido, ejecuta **step_two**; si no tiene nada que informar, ejecuta **step_three**; si tiene un espacio libre adelante, ejecuta **step_four**; en otro caso, i.e. tiene una muralla adelante, ejecuta **step_six**.

2. **step_two**: El robot inicializa en cero el valor del registro *robot_turns* y continúa la ejecución de **step_three**.

3. **step_three**: El robot examina lo que tiene al frente y termina la ejecución de la *ISR*.

4. **step_four**: El robot revisa si ha girado dos veces. Si es el caso (i.e. está apuntando en sentido opuesto), ejecuta **step_six**; en otro caso, continúa la ejecución de **step_five**.

5. **step_five**: El robot reinicia su contador de giros; avanza hacia adelante y termina la ejecución de la *ISR*.

6. **step_six**: El robot aumenta en una unidad su contador de giros; realiza un giro para evitar una muralla o para evitar retroceder y termina la ejecución de la *ISR*.

(b) Para evitar que las plantas de interior de su hogar se sequen, usted implementa un sistema de regado automatizado conectado a un mini-computador con ISA RISC-V. A cada una de sus plantas le añade un sensor de humedad y un regador, cuyos registros de 32 bits son mapeados en memoria desde la dirección 0xABADBABE:

Offset	Nombre	Descripción
0x00	`humidity_sensor_cactus`	Sensor humedad cactus
0x04	`humidity_sensor_monstera`	Sensor humedad monstera
0x08	`irrigator_status_cactus`	Registro estado regador cactus
0x0C	`irrigator_status_monstera`	Registro estado regador monstera
0x10	`irrigator_command_cactus`	Registro comando regador cactus
0x14	`irrigator_command_monstera`	Registro comando regador monstera

Nombre	Descripción	Valor
`humidity_sensor_*`	Porcentaje humedad	0-100
`irrigator_status_*`	Apagado	0
`irrigator_status_*`	Encendido	1
`irrigator_status_*`	Riego suave	2
`irrigator_status_*`	Riego medio	4
`irrigator_status_*`	Riego alto	8
`irrigator_command_*`	Encender	1
`irrigator_command_*`	Apagar	255
`irrigator_command_*`	Detener riego	128
`irrigator_command_*`	Regar (por defecto suave)	2
`irrigator_command_*`	Incrementar riego	3
`irrigator_command_*`	Decrementar riego	4

Luego de varias pruebas, usted desarrolla el siguiente programa para llevar a cabo el riego:

```
irrigation_system:
  li t0, 0xABADBABE            # Carga en el registro t0 el valor del literal 0xABADBABE
  li s0, 15
  li s1, 20
  li s2, 0
  li s3, 1
  step_one:
    lw t1, 8(t0)
    beq t1, s2, step_two
    bne t1, s3, step_three
    beq zero, zero, step_four                # Salto incondicional a step_four
  step_two:
    li t2, 1
    sw t2, 16(t0)
    beq zero, zero, step_four                # Salto incondicional a step_four
  step_three:
    li t2, 128
    sw t2, 16(t0)
  step_four:
    li t2, 2
    sw t2, 16(t0)
    li t2, 3
    sw t2, 16(t0)
    while_one:
        lw t3, 0(t0)
        blt t3, s0, while_one
    li t2, 4
    sw t2, 16(t0)
    while_two:
        lw t3, 0(t0)
        blt t3, s1, while_two
    li t2, 128
    sw t2, 16(t0)
    li t2, 255
    sw t2, 16(t0)
  end:                         # Término del programa.
```

En base a este programa, conteste las siguientes preguntas:

1. ¿Qué modo de comunicación existe entre el computador y el sistema de riego?

> **Solución:** El modo de comunicación existente consiste en *polling*, dado que en el *label* `step_four` se evidencia la consulta constante sobre el valor del sensor de humedad para gestionar el riego de la planta. No obstante lo anterior, también se aceptan las siguientes respuestas:
>
> - Comunicación en base a interrupciones, si se justifica que el código anterior consiste en la ISR ejecutada por el computador para llevar a cabo el riego de una planta.

- *Memory mapped* I/O. Si bien esto alude a la forma en la que se accede a los datos de los registros asociados a los dispositivos, dada la ambigüedad del enunciado se acepta también como respuesta **siempre** que se señale como argumento una de dos opciones: (1) la única forma de acceso a dispositivos a RISC-V; (2) el acceso a los registros de los dispositivos con instrucciones de lectura y escritura de memoria.

2. 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 línea de código.

Solución: A continuación, se describe lo que hace el programa en cada **step**:

- `step_one`: Se revisa el estado del regador del cactus. Si está apagado, se ejecuta el código del *label* `step_two`. Si está regando, independiente de la intensidad, se ejecuta el código del *label* `step_three`. En cualquier otro caso (*i.e.* está encendido), se ejecuta el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores.

- `step_two`: Se enciende el regador del cactus y se procede a ejecutar el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores.

- `step_three`: Se detiene el riego del regador del cactus y se procede a ejecutar el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores. En este caso, no es necesario explicitar la ejecución de `step_four`.

- `step_four`: Se activa el riego del regador del cactus y se incrementa para que realice riego medio. Luego, en `while_one` se consulta el valor del sensor de humedad del cactus constantemente hasta que sea mayor o igual a un 15 %. Posteriormente, se decrementa el nivel de riego para que tenga una intensidad suave. Finalmente, en `while_two` se consulta el valor del sensor de humedad del cactus hasta que sea mayor o igual a un 20 %, momento en el que se detiene el regador y se apaga. Se otorgan **1.5 ptos.** si se describe correctamente; y se descuentan **0.5 ptos.** por cada detalle faltante en la descripción.

Pregunta 3: I/O y RISC-V

Una empresa de instrumentalización ambiental le ha solicitado realizar la interfaz análogo-digital para un pluviómetro.

El elemento electromecánico del pluviometro es un botón que es presionado brevemente por un balancín que alterna al llenarse de agua en uno de sus extremos. El volumen de llenado es conocido, por lo que al contar la cantidad de veces que es presionado el botón podemos saber la cantidad de agua caída.

Como componente de control se le entrega un microcontrolador FU540-C000 de SiFive, que implementa la ISA de RISC-V. Dicho microcontrolador tiene puertos de entrada y salida digitales llamados GPIO (General Purpose Input/Ouput), cuyos registros de estado y control son de 32 bits y se encuentran mapeados en memoria desde la dirección 0x10060000, son los siguientes:

Offset	Name	Description
0x00	input_val	Contiene el valor de entrada
0x04	input_en	Controla si está habilitado como una entrada
0x08	output_en	Controla si está habilitado como una salida
0x0C	output_val	Controla el valor de salida
0x18	rise_ie	Controla si está habilitada las interrupciones por flanco de subida
0x20	fall_ie	Controla si está habilitada las interrupciones por flanco de bajada

Cada uno de los 32 bits de estos registros controla independientemente una de de 32 señales digitales de entrada o salida. De las cuales en este microcontrolador tiene disponible solo las 16 menos significativas, GPIOs del 0 al 15.

Su objetivo será detectar cuándo se acciona el botón, esto por medio de una interrupción por flanco de subida asociada al GPIO, e incrementar una variable global de 32 bits cada vez que esto ocurra. Leerá la señal proveniente del botón utilizando el GPIO 15.

(a) Explique qué problema tendrá con las características de la señal proveniente del botón y qué componente debería poner entre el botón y microcontrolador para obtener el resultado esperado.

Solución: Al apretar el botón la señal no pasará de 0 a 1 inmediatamente, oscilará alrededor del voltaje asociado al 1 lógico antes de estabilizarse. Para solucionar esto podemos poner un Debouncer que estabilizará la señal.

(b) Escriba una subrutina que configure el GPIO 15 como una entrada con la interrupción por flanco de subida habilitada, e inicialice en 0 la variable global del contador. Respete el estándar de llamadas de RISC-V.

Solución:

```
.data
    contador .space 4 # variable global de 32 bits
.text
    subrutina:
        li t0, 0x10060000  #dirección base
        li t1, 0x00008000 # 16° bit = 1
        sw t1, 0x04(t0) # entrada
        sw zero, 0x08(t0) # salida
        sw t1, 0x018(t0) # int. por flanco de subida
        sw zero, 0x20(t0) # int. por flanco de bajada
        la t2, contador  # |
        sw zero, 0(t2)  # | contador = 0
        ret
```

(c) Escriba la ISR que se encargará de incrementar la variable global definida anteriormente. Asuma que esta ISR ya está referenciada en el vector de interrupciones correspondiente y utilice la instrucción mret para finalizarla.

Solución:

```
isr:
     addi sp, sp, -8
     sw t0, 0(sp)
     sw t1, 4(sp)
     la t0, contador
     lw t1, 0(t0)
     addi t1,t1,1
     sw t1, 0(t0)
     lw t1, 4(sp)
     lw t0, 0(sp)
     addi sp, sp, 8
     mret
```