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

IIC2343 - Arquitectura de Computadores (I/2023)

Tarea 3

Pauta de evaluación

Publicación: Lunes 15 de Mayo 10:00
Entrega: Viernes 2 de Junio 20:00

Consideraciones

- Respuestas sin desarrollo o justificación no tendrán puntaje.
- Cada pregunta podría tener más de un desarrollo válido. La pauta evalúa eso y este documento solo muestra una alternativa de solución.
- Cualquier detección de infracción al código de honor será sancionada.

Código de Honor de la UC

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

Pregunta 1: RISC-V (30 ptos.)

RISC-V es un *Instruction Set Architecture* (ISA) del tipo *Reduced Instruction Set Architecture* (RISC) que sigue un esquema *load-store* y, además, es *open source*, siendo mantenida por RISC-V International y propuesta originalmente por Krste Asanović y David Patterson en la Universidad de California, Berkeley.

Para poder trabajar con el assembly descrito por la especificación de RISC-V, usaremos el emulador RARS, que sólo requiere tener Java 8 o superior instalado para correr en sus computadores. Este emulador entrega acceso al set de instrucciones básico más las extensiones para punto flotante, palabras de largo variable, entre otras y algunas pseudo-instrucciones. Son libres de usar el emulador o plataforma que quieran, pero tengan en consideración que las correcciones se harán usando RARS 1.6.

(a) (4 ptos.) La especificación de RISC-V incluye una convención de llamada, que a partir de la versión V20191213 se encuentra definida como parte del psABI. Investigue y escriba brevemente qué indica la convención de llamada para la extensión base RV32I, los aspectos más relevantes de esta, cómo se utiliza y por qué es necesaria. Para las siguientes secciones de la tarea deberá escribir programas en *assembly* RISC-V, por lo que es importante que preste atención a cómo se utiliza la convención de llamada.

Solución: Los elementos mínimos que deben mencionar son los siguientes:

- Alias de todos los 32 registros principales y su uso esperado. Pueden o no mencionar los registros de punto flotante.
- Responsabilidad de respaldar `s*` por parte del *callee*.
- Responsabilidad de respaldar `t*` por parte del *caller*.

Distribución de puntaje

- **1 punto** por indicar el alias de los registros.
- **1 punto** por indicar el uso esperado de cada registro.
- **1 punto** por *callee backups*.
- **1 punto** por *caller backups*.

(b) (4 ptos.) Para esta pregunta, deberá desarrollar un programa que detecte si un número es primo gemelo. Un número primo p es gemelo si existe otro número primo q tal que $|p-q| = 2$. En este programa, recibirá en la sección `.data` una etiqueta `N`, que corresponde a un entero positivo. Al finalizar la ejecución, deberá usar el `ecall 1` de RARS para imprimir en consola un 1 si N es primo gemelo y un 0 en otro caso. La primera parte de su archivo se debiese ver de la siguiente forma:

```
.globl start
.data
    N: .word 11
.text
    start:
        # do stuff
```

Se evaluará la correctitud del código, que este respete la convención de llamada y que pueda aceptar un número N arbitrario como *input*.

Solución: Una posible solución que hace uso de la extensión F de RISC-V es la siguiente:

```
.globl start
.data
    N: .word 17
.text
    start:
        lw a1, N
        mv a0, a1
        call is_prime               # is_prime(N)
        beqz a0, end                # a0 == 0 if N is not prime
        addi a0, a1, -2
        call is_prime               # is_prime(N - 2)
        bgtz a0, end                # a0 > 0 if N is twin prime
        addi a0, a1, 2
        call is_prime               # is_prime(N + 2)
    end:                            # a0 == 1 if N is twin prime else a0 == 0
        li a7, 1
        ecall
        li a7, 10
        ecall
    is_prime:
        li t0, 1
        ble a0, t0, not_prime       # Edge case for N <= 1
        fcvt.s.w ft0, a0            # ft0 = float(a0)
        fsqrt.s ft0, ft0            # ft0 = sqrt_2(ft0)
        fcvt.w.s t0, ft0            # t0  = int(ft0)
        li t1, 2
        while:
            rem t2, a0, t1          # if a0 % t1 == 0 -> N not prime
            beqz t2, not_prime
            addi t1, t1, 1
            ble t1, t0, while
        prime:
            li a0, 1
            ret
        not_prime:
            li a0, 0
            ret
```

Distribución de puntaje

- **1 punto** por respetar la convención de llamada.
- **1.5 puntos** por identificar correctamente si un número es primo o no.
- **1 punto** por identificar si un número es un primo gemelo o no.
- **0.5 puntos** por utilizar la `ecall 1` para entregar su respuesta.

(c) (8 ptos.) Para esta pregunta, deberá programar una pequeña versión de juego FizzBuzz en RISC-V ASM. Para esto, recibirá en la sección `.data` una etiqueta `N`, que corresponde a un entero positivo y deberá escribir, a partir de la etiqueta `out`, todos los números entre 0 y `N` como `.word`, con la salvedad de que si el número es divisible por 3, este se reemplaza por 70 (ASCII "F"); en caso de ser divisible por 5, se reemplaza por 66 (ASCII "B"); y si es divisible por ambos, se reemplaza por 7066. Además, si un número es primo gemelo, deberá reemplazarlo por 80 (ASCII "P"). A modo de ejemplo, si $N = 15$ la secuencia que se guardará en `out` será: `7066, 1, 2, 80, 4, 80, 70, 80, 8, 70, 66, 80, 70, 80, 14, 7066`
La primera parte de su archivo se debiese ver de la siguiente forma:

```
.globl start
.data
    N: .word 10
    out: .word
.text
    start:
        # do stuff
```

Se evaluará la correctitud del código, que este respete la convención de llamada y que pueda aceptar un número N arbitrario como *input*.

Solución: Una posible solución es la siguiente:

```
.globl start
.data
    N: .word 15
.text
    start:
        lw s1, N                    # max number
        li s0, 0                    # counter
        li s2, 3                    # first divisor
        li s3, 5                    # second divisor
        while:
            mv a0, s0
            call is_twin_prime      # is_twin_prime(a0 = counter). No need to save t* registers
            bnez a0, twin_prime
            rem t0, s0, s2          # t0 = s0 % 3 = remainder of division by 3
            rem t1, s0, s3          # t1 = s0 % 5 = remainder of division by 5
            beq t1, t0, fizzbuzz    # t0 == t1 -> s0 divisible by 3 and 5? -> FizzBuzz
            beq t0, zero, fizz      # t0 == 0  -> s0 divisible by 3 -> Fizz
            beq t1, zero, buzz      # t1 == 0  -> s0 divisible by 5 -> Buzz
            fizzbuzz:
                bnez t1, skip       # t1 != 0  -> s0 not divisible by 3 nor 5
                li a0, 7066
                j continue
            fizz:
```

```
            li a0, 70
            j continue
        buzz:
            li a0, 66
            j continue
        twin_prime:
            li a0, 80
            j continue
        skip:
            mv a0, s0
        continue:
            mv a1, s0
            call store              # store(a0 = value_to_store, a1 = counter)
            addi s0, s0, 1
            bgt s0, s1, end         # If s0 > s1 -> counter > N, array is finished
            j while
store:
    addi sp, sp, -8                 # s* registers are callee-saved
    sw s0, 0(sp)
    sw s1, 4(sp)
    la s0, N                        # s0 = N variable address
    addi s0, s0, 4                  # s0 += 4 for first available array address
    li s1, 4                        # s1 = 4 (bytes stored for each array element)
    mul s1, s1, a1
    add s0, s1, s0                  # Stored value address = Address(N)+4+4*counter
    sw a0, 0(s0)                    # Value stored
    lw s0, 0(sp)                    # s* registers restored
    lw s1, 4(sp)
    addi sp, sp, 8
    ret
end:
    li a7, 10
    ecall
is_twin_prime:                      # a0 = number to check. a0=1 if twin, a0=0 elsewhen
    addi sp, sp, -4                 # We store ra for the final return.
    sw ra, 0(sp)
    mv a1, a0
    call is_prime                   # is_prime(a0=counter). a0=1 if prime, a0=0 elsewhen
    beqz a0, end_is_twin_prime      # a0 == 0 if counter is not prime
    addi a0, a1, -2
    call is_prime                   # is_prime(a0 - 2)
    bgtz a0, end_is_twin_prime      # a0 > 0 if counter is twin prime
    addi a0, a1, 2
    call is_prime                   # is_prime(a0 + 2)
    end_is_twin_prime:              # a0 == 1 if N is twin prime else a0 == 0
        lw ra, 0(sp)                # We restore ra for correct return
        addi sp, sp, 4
        ret
    is_prime:
        li t0, 1
        ble a0, t0, not_prime       # Edge case for counter <= 1
        fcvt.s.w ft0, a0            # ft0 = float(a0)
        fsqrt.s ft0, ft0            # ft0 = sqrt_2(ft0)
        fcvt.w.s t0, ft0            # t0  = int(ft0)
        li t1, 2
        while_is_prime_check:
            rem t2, a0, t1          # if a0 % t1 == 0 -> counter not prime
            beqz t2, not_prime
            addi t1, t1, 1
            ble t1, t0, while_is_prime_check
    prime:
        li a0, 1
        ret
    not_prime:
        li a0, 0
        ret
```

Distribución de puntaje

- **1 punto** por respetar la convención de llamada.
- **1 punto** por indicar cuando es `fizz (70, 0x0046)` correctamente.
- **1 punto** por indicar cuando es `buzz (66, 0x0042)` correctamente.
- **1 punto** por indicar cuando es `fizzbuzz (7066, 0x1b9a)` correctamente.
- **2 puntos** por indicar cuando es un primo gemelo `(80, 0x50)` correctamente.
- **2 puntos** por resultado correcto para `N=2,10,15,43`. Se descuenta **1 punto** si no se guarda el resultado en memoria.

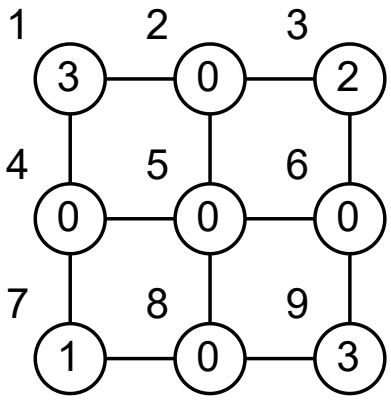
(d) (14 ptos.) Para esta pregunta, deberá resolver un puzzle utilizando RISC-V ASM. Recibirá como *input* un entero `N` y dos arreglos, `M` y `T`. `N` representa la cantidad de nodos de un grafo; `M` representa su matriz de adyacencia; y `T` representa el tipo de cada uno de sus nodos. Existen cuatro tipos de nodo:

- `Inicio` = `1`
- `Final` = `2`
- `Punto` = `3`
- `Vacío` = `0`

El objetivo del puzzle es trazar un camino desde `Inicio` hasta `Final`, conectado a todos los nodos `Punto`. El camino no puede pasar dos veces por el mismo vértice ni borde.

Como resultado, su programa deberá usar el `ecall 1` de RARS para imprimir, en orden, los vértices que recorre el camino de la solución.

A modo de ejemplo, un posible puzzle junto con su matriz de adyacencia, lista de tipo y solución, podría ser el siguiente:



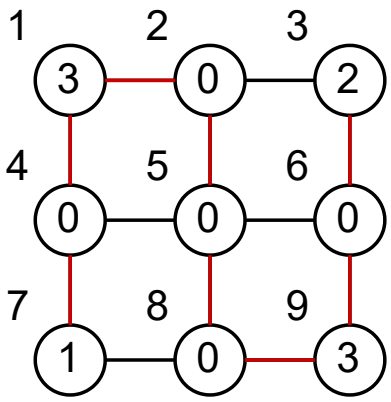
Su matriz de adyacencia es:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Y su lista de tipos corresponde a:

$$3, 0, 2, 0, 0, 0, 1, 0, 3$$

La solución sería:



$$7, 4, 1, 2, 5, 8, 9, 6, 3$$

El *input* asociado podría verse de la siguiente manera:

```
.globl start
.data
    # --- No modificar labels ---
    N: .word 9
    M: .word 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
     0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
      0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0
    T: .word 3, 0, 2, 0, 0, 0, 1, 0, 3
    # --- End no modificar labels---
    # de aca para abajo van sus variables en memoria
.text
    start:
        # do stuff
```

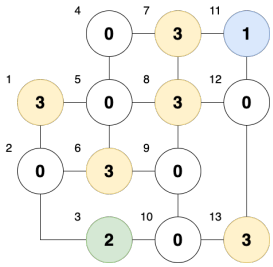
Se evaluará que se respete la convención de llamada y que se entregue un *output* correcto a partir de una serie de casos de prueba.

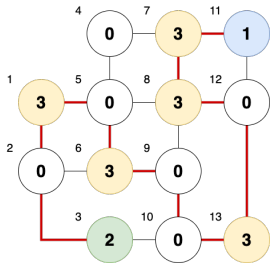
Si lo desea, puede optar a un máximo de **6 ptos.** si realiza un programa que solo entregue un camino válido entre el nodo `Inicio` y el nodo `Final` del grafo entregado, sin necesidad de pasar por todos los nodos `Punto`.

Solución: Para este programa, se evalúa a partir de los siguientes dos puzzles:

Puzzle 1 y su solución

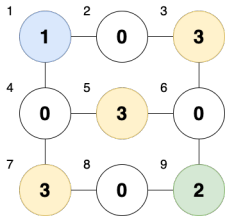
```
.globl start
.data
N: .word 13
M: .word 0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,
         1,0,1,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,
         0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,
         1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0
T: .word 3,0,2,0,0,3,3,3,0,0,1,0,3
# solution S: 11,7,8,12,13,10,9,6,5,1,2,3
```

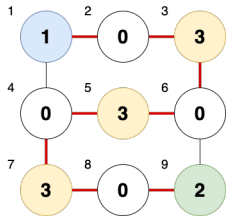




Puzzle 2 y su solución

```
.globl start
.data
N: .word 9
M: .word 0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,
         0,0,0,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0
T: .word 1,0,3,0,3,0,3,0,2
# solution S: 1,2,3,6,5,4,7,8,9
```





Distribución de puntaje

- Si realiza la implementación encontrando el camino correcto:
  - **14 puntos** si pasa los dos *tests*.
  - **10 puntos** si pasa al menos un *test*.
- Si realiza la implementación encontrando un camino válido:
  - **6 puntos** si pasa al menos uno de los dos *tests*.
- Si los resultados de los *tests* son incorrectos en cualquiera de los dos casos anteriores:
  - **3 puntos** si el programa compila, respeta la convención de llamada y al menos entrega un camino (*i.e.* trata de buscar una solución).

Pregunta 2: Input/Output (30 ptos.)

(a) (15 ptos.) Luego del accidente de East Plestine, Ohio, se decide actualizar los sistemas de monitoreo de borde de vía para trenes (WTMS), que consisten en un pequeño computador instalado al borde de una vía con una serie de sensores conectados que miden diferentes parámetros de operación de un tren (temperatura de los ejes, sobredimensión o protuberancias, derrame de líquidos, etc.) y transmiten esta información por radio a la locomotora, junto con almacenar un reporte. Adicionalmente, el sistema se puede actualizar agregando nuevos sensores o componentes a lo largo de su vida útil.

Asuma que existe un componente de control que implementa la ISA de RISC-V. Este componente, correspondiente a un microcontrolador, tiene puertos de entrada y salida digitales llamados GPIO (*General Purpose Input/Ouput*), cuyos registros de estado y comando son de 32 bits y se encuentran mapeados en memoria desde la dirección 0x10030000. Estos son:

Offset	Nombre	Descripción
0x00	`dir_ise`	Contiene la dirección ISR del manejo de alerta del tren
0x04	`reg_tr`	Registro de comandos del tren
0x08	`sensor_te`	Registro de estado de sensor de temperatura
0x0C	`sensor_pr`	Registro de estado de sensor de protuberancia
0x18	`sensor_dr`	Registro de estado de sensor de derrame
0x20	`rad_cl`	Registro de comandos de la radio

A su vez, las especificaciones de estado o comando por registro se definen en la siguiente tabla:

Nombre	Comando o Estado	Valor
`reg_tr`	Notificar locomotora	255
`reg_tr`	Freno de emergencia	127
`sensor_te`	Temperatura alta	255
`sensor_te`	Temperatura baja	1
`sensor_te`	Temperatura normal	127
`sensor_pr`	Sobredimensión del tren o protuberancias	4
`sensor_pr`	Rieles libres de obstáculos	2
`sensor_dr`	Existencia de derrame	3
`sensor_dr`	Sin derrame de fluidos en el tren	1
`rad_cl`	Mandar reporte de sensores	1

Escriba la ISR que se encargará de apagar activar el freno de emergencia, notificar a la locomotora y mandar un reporte cuando un estado sea no deseado. Un estado no deseado se define cuando tiene la combinación de uno o más de los siguientes estados: derrame; temperatura distinta a la normal; existencia de elementos sobresalientes. Por políticas del seguro de calidad del sistema, es requerido que la interacción con cada registro de estado de cualquier sensor se haga a través de una subrutina dentro de la ISR, respetando el estándar de llamadas de RISC-V. Asuma que esta ISR ya está referenciada en el vector de interrupciones correspondiente y utilice la instrucción `mret` de RISC-V para finalizarla.

Solución: Una solución podría ser la siguiente:

```
wtms_isr:
    addi sp, sp, -16        # s* registers are callee saved
    sw s0, 0(sp)
    sw s1, 4(sp)
    sw s2, 8(sp)
    sw s3, 12(sp)
    call sens_te_sub
    mv s0, a0               # s0 = sensor_te state
    call sens_pr_sub
    mv s1, a0               # s1 = sensor_pr state
    call sens_dr_sub
    mv s2, a0               # s2 = sensor_dr state
    li s3, 0                # s3 = number of unwanted states
    li t0, 127              # Check temperature
    beq s0, t0, norm_temp
    addi s3, s3, 1          # Unwanted count += 1
    norm_temp:
        li t0, 2            # Check protuberances
        beq s1, t0, norm_pr
        addi s3, s3, 1      # Unwanted count += 1
    norm_pr:
        li t0, 1
        beq s2, t0, norm_dr
        addi s3, s3, 1      # Unwanted count += 1
    norm_dr:                # Check if all sensors are normal
        li t0, 2
        bgtz s3, emergency_action
        j subend
    emergency_action:       # Emergency if unwanted count > 0
        call rad_cl_sub
        call reg_tr_sub
        j subend
    sens_te_sub:
        li t0, 0x10030008   # t0 = sensor_te address
        lw a0, 0(t0)        # a0 = sensor_te value
        ret
    sens_pr_sub:
        li t0, 0x1003000C   # t0 = sensor_pr address
        lw a0, 0(t0)        # a0 = sensor_pr value
        ret
    sens_dr_sub:
        li t0, 0x10030018   # t0 = sensor_dr address
        lw a0, 0(t0)        # a0 = sensor_dr value
        ret
    reg_tr_sub:
        li t0, 0x10030004   # t0 = reg_tr address
        li t1, 255          # t1 = notify locomotive command
        sw t1, 0(t0)
        li t1, 127          # t1 = emergency brake command
        sw t1, 0(t0)
        ret
    rad_cl_sub:
        la t0, 0x10030020   # t0 = rad_cl address
        li t1, 1            # t1 = send sensors report command
        sw t1, 0(t0)
        ret
    subend:
        lw s0, 0(sp)        # s* registers restored
        lw s1, 4(sp)
        lw s2, 8(sp)
        lw s3, 12(sp)
        addi sp, sp, 16
        mret                # ISR ret
```

Distribución de puntaje

- **1 punto** por respetar la convención de llamada.
- **2 puntos** por sintaxis y estructura correcta de la subrutina.
- **2 puntos** por interactuar correctamente con `reg_tr` a través de una subrutina.
- **2 puntos** por interactuar correctamente con `sensor_te` a través de una subrutina.
- **2 puntos** por interactuar correctamente con `sensor_pr` a través de una subrutina.
- **2 puntos** por interactuar correctamente con `sensor_dr` a través de una subrutina.
- **2 puntos** por interactuar correctamente con `rad_cl` a través de una subrutina.
- **2 puntos** por detectar correctamente cuando hay una situación no deseada.

Se descuentan **3 puntos** si hay más de un registro con el que no se interactúe a través de una subrutina.

(b) (15 ptos.) La librería `fakeio`, escrita en Python 3.10, entrega una clase que simula un controlador de dispositivos de I/O *Port-Mapped*. La librería viene acompañada de una pequeña documentación de uso, pero a grandes rasgos expone una interfaz `OUT` que toma como argumentos un puerto, un valor y no retorna nada; e `IN`, que toma como argumento solo un número de puerto y retorna un *string*. Los dispositivos disponibles son:

- Una cinta magnética, `tape`.
- Un generador de enteros aleatorios, `entropy`.
- Una impresora, `printer`.
- Un coprocesador de números de punto flotante, `coprocessor`.

Deberá elaborar un pequeño programa en Python que haga uso **exclusivo** de esta librería y que realice las siguientes interacciones:

1. Leer un par de caracteres de la cinta magnética a partir de una posición `data` obtenida del dispositivo `entropy`.
2. Realizar tres operaciones diferentes en el coprocesador usando como operandos los caracteres obtenidos de la cinta magnética, interpretados como su valor ASCII.
3. Escribir el resultado de la operación al inicio de la cinta magnética.
4. Escribir tres caracteres de la cinta magnética en la impresora y terminar con un salto de línea.

La librería `fakeio` se encuentra en el siguiente enlace, junto con su documentación. Esta librererería necestia Python 3.10 o superior para funcionar.

Si encuentra algún *bug*, puede reportarlo en las *issues* del repositorio para arreglar su funcionamiento.

Solución: Una posible solución es la siguiente:

```python
from fakeio import PMIO

if __name__ == '__main__':
    pmio = PMIO()
    # Entropy interaction
    pos1 = pmio.IN(1)
    pos2 = pmio.IN(1)
    # Tape interactions
    pmio.OUT(5, int(pos1))              # Set head to first position
    pmio.OUT(4, 1)                      # Read 1 byte
    char1 = ord(pmio.IN(2))             # First char reading
    pmio.OUT(5, int(pos2))              # Set head to second position
    pmio.OUT(4, 1)                      # Read 1 byte
    char2 = ord(pmio.IN(2))             # Second char reading
    # Coprocessor interactions
    pmio.OUT(6, char1)                  # First operand
    pmio.OUT(7, char2)                  # Second operand
    pmio.OUT(8, 0)                      # char1 + char2
    out = pmio.IN(9)
    pmio.OUT(8, 1)                      # (char1 + char2) - char2 = char1
    out = pmio.IN(9)
    pmio.OUT(8, 2)                      # char1 * char2
    out = pmio.IN(9)
    pmio.OUT(8, 3)                      # (char1 * char2) / char2 = char1
    out = pmio.IN(9)                    # out = char1
    # Tape interactions
    pmio.OUT(5, 0)                      # Set tape head to beginning
    pmio.OUT(3, int(float(out)) % 127)  # Use valid ASCII range and write it
    pmio.OUT(5, int(float(out)))        # Use result to set next head
    pmio.OUT(4, 3)                      # Read 3 bytes
    chrs = pmio.IN(2)                   # Get chars read
    # Printer interactions
    pmio.OUT(0, ord(chrs[0]))           # Print first char
    pmio.OUT(0, ord(chrs[1]))           # Print second char
    pmio.OUT(0, ord(chrs[2]))           # Print third char
    pmio.OUT(0, ord("\n"))              # Print newline
```

Distribución de puntaje

- **3 puntos** por interactuar correctamente con la librería, usando solo las interfaces `IN` y `OUT` y no con las clases de los dispositivos.
- **4 puntos** por interactuar con los dispositivos I/O en el orden esperado.
- **2 puntos** por interactuar correctamente con `entropy`.
- **4 puntos** por interactuar correctamente con `tape`.
- **4 puntos** por interactuar correctamente con `coprocessor`.
- **2 puntos** por interactuar correctamente con `printer`.

Notar que se evalúa a nivel de interacción y no según lo explicitado en el enunciado por la existencia de más de una versión de este, por su libre interpretación y porque el foco de esta pregunta se encuentra en interactuar de forma coherente con los dispositivos a través de instrucciones *Port-Mapped* I/O.