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

IIC2343 - Arquitectura de Computadores (II/2024)

Interrogación 2 - RISC-V

Instrucciones

Lea atentamente el enunciado. Tendrá hasta las 21:00 del 29 de Mayo para subirlo a Canvas. Cualquier duda que tenga, debe realizarla a través de las *issues* del repositorio del curso usando el *label* "`Actividad RISC-V`".

Pregunta 4: Batalla Pokémon (8 ptos.)

El Dr. Yadran quiere completar una Pokédex regional de Santiago, por lo que requiere de tu ayuda para completar su objetivo. Para ello, deberás simular el combate de los Pokémon^{TM} *starters* de Santiago haciendo uso del lenguaje de ensamblado RISC-V en el emulador RARS.

Cada Pokémon estará ubicado en la sección de `.data` y tendrá los siguientes valores almacenados de manera **consecutiva**.

Offset	Nombre	Descripción
0	**ID**	Número del Pokémon en la Pokédex
4	**Tipo**	Puede ser tipo: planta (0), fuego (1) o agua (2)
8	**Nivel**	Nivel del Pokémon
12	**EXP**	Puntos de experiencia. El Pokémon los obtiene al ganar peleas
16	**Estado**	Estado de salud del Pokémon. Saludable (0), envenenado (1), dormido (2) o desmayado (3)
20	**HP**	*Health Points*. Cantidad de puntos de salud que tiene el Pokémon
24	**HP total**	Cantidad máxima de HP que puede tener el Pokémon
28	**Ataque**	Puntos de ataque físico del Pokémon
32	**Defensa**	Puntos de defensa físico del Pokémon
36	**Ataque Especial**	Puntos de ataque especial del Pokémon
40	**Defensa Especial**	Puntos de defensa especial del Pokémon
44	**Velocidad**	Puntos de velocidad del Pokémon

La batalla va a consistir en una de enfrentamientos entre dos Pokémon. Se enfrentarán hasta que alguno de ellos llegue al **nivel** 40. Después de cada enfrentamiento, todos los Pokémon recuperarán completamente sus puntos de salud. De esta manera, el flujo de batalla general es el siguiente:

1. El Pokémon más rápido es el primero en atacar a su contrincante. Esto ocurre solo si **no está dormido**.
2. Luego, es el turno del Pokémon más lento para atacar. Esto ocurre solo si **no está dormido**.
3. Si alguno de los Pokémon pierde todos sus puntos de vida después de ser atacado, la pelea finaliza y el ganador se lleva los **puntos de experiencia**. El Pokémon que pierde pasa a estado **desmayado**.
4. Si ninguno de los Pokémon se desmayó en el turno, se revisa lo siguiente:
   - Si un Pokémon se encuentra **envenenado**, pierde puntos de salud después de la ronda.
   - Si un Pokémon está **dormido**, tiene 50 % de probabilidades de despertar.
5. Al finalizar el turno, hay un 10 % de probabilidades que un Pokémon se **envenene** y un 10 % de probabilidades de que se **duerma**.
6. El flujo anterior se repite hasta que haya un ganador.

Este flujo **ya se encuentra implementado** en un archivo base adjunto a este enunciado, **¡no debe programarlo!** Su tarea consistirá en **completar** el código de un conjunto de subrutinas, las que son descritas en las siguientes secciones. Adicionalmente, en este mismo archivo contará con subrutinas complementarias **ya implementadas** para facilitar su desarrollo.

1-. Ataque: Subrutina `attack`

Para computar el ataque de un Pokémon en un turno, debe hacer uso de la siguiente fórmula:

$$\texttt{Poder Ataque} = \texttt{Ataque físico (atacante)} - \texttt{Defensa fisica (defensor)} + \texttt{Ponderador tipo} + 4 \times \texttt{Ataque crítico}$$

El `Ponderador tipo` corresponde a un ponderador que indica si el ataque es más efectivo (o no) dado el tipo de los Pokémon atacante y defensor.

Pokémon Atacante	Pokémon Defensor	Ponderador
Planta (0)	Planta (0)	0
Planta (0)	Fuego (1)	-3
Planta (0)	Agua (2)	3
Fuego (1)	Planta (0)	3
Fuego (1)	Fuego (1)	0
Fuego (1)	Agua (2)	-3
Agua (2)	Planta (0)	-3
Agua (2)	Fuego (1)	3
Agua (2)	Agua (2)	0

Para lograr lo anterior, debe implementar la subrutina `attack`. Esta debe tomar como argumentos las direcciones del primer y segundo Pokémon a través de los registros `a0` y `a1` , respectivamente, y hacer lo siguiente:

1. Primero, debe verificar si el Pokémon se encuentra **dormido**. Si lo está, entonces se retorna y pierde su turno de ataque.
2. Si no esta dormido, entonces puede atacar. Para calcular el ataque, debe obtener el ponderador por tipo; el ataque físico del atacante; la defensa física del defensor; y determinar si el ataque es crítico.
3. Para obtener el ponderador por tipo, se debe llamar a la subrutina `_type_boost`. Esta toma como argumentos el valor de la dirección base del Pokémon atacante en el registro `a0` , y en el registro `a1` el valor de la dirección base del Pokémon defensor. La subrutina retorna el ponderador (-3, 0 o 3 según sea el caso) en el registro `a0` .
4. Para saber si el ataque es crítico, se debe llamar a la subrutina `_critical_damage`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` y hace lo siguiente: Retorna 1 con una probabilidad del 10 %, y 0 en cualquier otro caso. El valor de retorno es almacenado en `a0` . Si el valor de retorno es 1, entonces el daño es crítico y el daño del Pokémon atacante aumenta en 4 puntos. Adicionalmente imprime en consola el nombre del Pokémon , indicando si realizará un daño crítico o no.
5. Luego de obtener los valores anteriores, computa el valor de ataque según la fórmula antes descrita. Si el valor calculado es menor a 0, entonces el ataque será igual a 0.
6. Se restan los puntos calculados al **HP** del Pokémon defensor.
7. Si el **HP** del Pokémon defensor es mayor a 0, se debe retornar.
8. Si el **HP** del Pokémon defensor es menor o igual a 0, el valor de **HP** se deja en 0 y se debe llamar a la subrutina `_faint`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` e imprime en consola su nombre, indicando que se ha desmayado.
9. Retorna a donde fue llamada.

2. Turno ataque: Subrutina `turn_attack`

Para llevar acabo el ataque por turnos, se debe implementar la subrutina `turn_attack`. Esta debe tomar como argumentos las direcciones del primer y segundo Pokémon a través de los registros `a0` y `a1` , respectivamente, y hacer lo siguiente:

1. Primero se debe decidir el orden en que deben atacar. El Pokémon con mayor **velocidad** es el que debe golpear primero.
2. El Pokémon con mayor velocidad ataca primero. Para atacar se debe llamar a la subrutina `attack`. Se debe guardar la dirección base del Pokémon más rápido en el registro `a0` y la dirección base del Pokémon más lento se debe guardar en el registro `a1` .
3. Si al retornar de `attack` el Pokémon que fue atacado tiene **HP** igual a 0, se debe retornar (pierde el turno de atacar).
4. Ahora es el turno del Pokémon con menor velocidad. Para atacar se debe llamar a la subrutina `attack`. Se debe guardar la dirección base del Pokémon más lento en el registro `a0` y la dirección base del Pokémon más rápido se debe guardar en el registro `a1` .
5. Retorna a donde fue llamada.

3. Experiencia: Subrutina `experience`

Cada vez que un Pokémon gana una batalla, recibe puntos de experiencia (**EXP**). La experiencia que gana un Pokémon depende de: el nivel que se encuentra; su ataque físico y su ataque especial. La fórmula con la que se calcula la **experiencia** recibida por ganar una batalla es la siguiente:

$$\texttt{Experiencia ganada} = \texttt{Nivel} + \texttt{Ataque físico} + \texttt{Ataque especial}$$

Para esto, se debe implementar la subrutina `experience`. Esta debe tomar como argumento el valor de la dirección base del Pokémon a través del registro `a0` y hacer lo siguiente:

1. Calcular la experiencia que gana un Pokémon al ganar una batalla a través de la fórmula antes descrita.
2. Sumar el valor calculado a la **experiencia** que el Pokémon ya posee.
3. Si la **experiencia** total es menor a 100, se debe retornar.
4. Si la **experiencia** total es mayor o igual a 100, se debe subir de nivel. Antes de hacerlo, debe guardar como **experiencia** la resta entre el valor computado y 100. Esto es importante, ya que este parámetro siempre debe estar entre 0 y 99.
5. Para subir el nivel del Pokémon , se debe llamar a la subrutina `_level_up`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` y actualiza las estadísticas del mismo: aumenta el nivel, ataque físico, ataque especial, defensa física, defensa especial, velocidad y, por último, llama a la subrutina `evolve` descrita en la siguiente sección. Adicionalmente, imprime en consola su nombre indicando que ha subido de nivel.
6. Retorna a donde fue llamada.

4. Evolución: Subrutina `evolve`

Dada la naturaleza de cada Pokémon , al hacerse más fuertes por su experiencia pueden transformarse en una nueva especie más poderosa, lo que se conoce como **evolución**. Por lo anterior, cada vez que un Pokémon sube de nivel, se verifica si este debe evolucionar. Para esto, se debe implementar la subrutina `evolve`. Esta debe tomar como argumento el valor de la dirección base del Pokémon a través del registro `a0` y hacer lo siguiente:

1. Debe buscar en `evolutions`, correspondiente a una lista de tripletas en `.data`, la tripleta del Pokémon recibido como argumento. Cada tripleta tiene la siguiente estructura: ID del Pokémon, ID del Pokémon al que evoluciona, y **nivel** en el que evoluciona.
2. Si **no** se encuentra una tripleta con el primer elemento igual al **ID** del Pokémon recibido como argumento, se debe retornar.
3. Si se encuentra una tripleta con el primer elemento igual al **ID** del Pokémon recibido como argumento, se debe que verificar si tiene el nivel necesario para evolucionar o no.
4. Para verificar si tiene el nivel necesario para evolucionar, este debe tener un nivel mayor o igual al tercer elemento de la tripleta. Si es menor, se debe retornar.

5. Si el Pokémon cumple con los requisitos para evolucionar, se debe cambiar su **ID** al segundo valor contenido en la tripleta y llamar a la subrutina `_print_evolve_text`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` e imprime en consola su nombre, indicando que evolucionó.
6. Retorna a donde fue llamada.

Bonus opcional - 5. Estado: Envenenado

Al final de cada turno, si un Pokémon tiene como estado de salud **envenenado**, su vida (**HP**) debe decrementar en 2 puntos. Para esto, se debe implementar la subrutina `poison_damage`. Esta debe tomar como argumento el valor de la dirección base del Pokémon a través del registro `a0` y hacer lo siguiente:

1. Primero debe verificar si el Pokémon tiene como estado de salud **envenenado**. Si no, debe retornar.
2. Si el Pokémon esta **envenenado**, se debe restar 2 puntos a su salud **HP**.
3. Se debe llamar a la subrutina `_print_poison_text`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` e imprime en consola el nombre del Pokémon **envenenado**.
4. Si los puntos de vida disminuyen a un valor menor o igual a 0, se debe guardar un 0 en su **HP** y llamar a la subrutina `_faint`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` e imprime en consola su nombre, indicando que se desmayó.
5. Retorna a donde fue llamada.

Bonus opcional - 6. Estado: Dormido

Al final de cada turno, si un Pokémon tiene como estado de salud **dormido**, tiene un 50 % de probabilidades de despertar.

Para esto, se debe implementar la subrutina `wake_up`. Esta debe tomar como argumento el valor de la dirección base del Pokémon a través del registro `a0` y hacer lo siguiente:

1. Primero, debe verificar si el Pokémon tiene como estado de salud **dormido**. Si no, debe retornar.
2. Llamar a la subrutina `_rand_number`. Esta recibe como argumento un número a través del registro `a0` , y retorna un valor en el rango [0,`a0` [. Por ejemplo, si `a0` = 10, entonces retorna un número entre 0 y 9. El valor retornado se almacena en el registro `a0` .
3. Dado el número generado, se decide con un 50 % de probabilidad si el Pokémon despierta o no. Si sigue **dormido**, hay que retornar.
4. Si el Pokémon despierta, se debe cambiar su **estado** a **saludable** y llamar a la subrutina `_print_wake_up_text`. Esta toma como argumento la dirección base del Pokémon en el registro `a0` e imprime en consola el nombre su nombre, indicando que despertó.
5. Retorna a donde fue llamada.

Importante - Convención RISC-V

En esta actividad, no se evaluará que respete la convención RISC-V. No obstante, si no se asegura del respaldo de los registros en las subrutinas a completar, **podría sobreescribir valores que impliquen una implementación incorrecta**. Por lo tanto, trate de respetar la convención revisando el código base entregado y asegurando el respaldo de los registros cuando corresponda.

Puntajes

El puntaje se otorga por el funcionamiento correcto de cada subrutina. Si presenta como máximo un error de implementación, se otorga la mitad del puntaje. En otro caso, se otorgan **0 ptos.**
El puntaje de bonificación, en caso de obtenerlo, se sumará directamente al total de la I2.

(a) (1.5 ptos.) Implementa la subrutina `attack` según lo especificado en Ataque.

Solución:

```
attack:                                 # a0: direccion del pokemon a atacante. a1: direccion del
    pokemon a defensor.
        addi sp, sp, -16
        sw ra, 0(sp)
        sw a0, 4(sp)
        sw a1, 8(sp)
        sw s0, 12(sp)

        lw t0, 16(a0)
        addi t1, zero, 2
        bne t0, t1, no_dormido
        addi sp, sp, 16
        jalr zero, 0(ra)

        no_dormido:
        jal ra, _type_boost
        add s0, zero, a0
        lw a0, 4(sp)
        jal ra, _critical_damage
        addi t0, zero, 4
        mul t0, a0, t0
        add s0, s0, t0

        lw a0, 4(sp)
        lw a1, 8(sp)
        lw t0, 28(a0) # Se guarda en t0 el ataque
        lw t1, 32(a1) # Se guarda en t1 la defensa del segundo
        sub t0, t0, t1 # Ataque - Defensa
        add s0, s0, t0 # (Ataque - Defensa) + critico + ponderador
        bgt s0, zero, ataque_positivo
        add s0, zero, zero

        ataque_positivo:
        lw t0, 20(a1) # Se guarda en t1 el HP del segundo
        sub t0, t0, s0
        bgt t0, zero, seguir_ataque
        sw zero, 20(a1)
        lw a0, 8(sp)
```

```
        jal ra, _faint
        lw a0, 4(sp)
        lw ra, 0(sp)
        lw s0, 12(sp)
        addi sp, sp, 16
        jalr zero, 0(ra)

        seguir_ataque:
        sw t0, 20(a1) # t1 se guarda en el HP del segundo
        lw ra, 0(sp)
        lw s0, 12(sp)
        addi sp, sp, 16
        jalr zero, 0(ra)
```

1. (0.25 puntos) El *output* del test `test_01_attack.asm` es corecto.
2. (0.25 puntos) El *output* del test `test_02_attack.asm` es corecto.
3. (0.25 puntos) El *output* del test `test_03_attack.asm` es corecto.
4. (0.25 puntos) El *output* del test `test_04_attack.asm` es corecto.
5. (0.5 puntos) El *output* del test `test_05_attack_sp.asm` es corecto.

(b) (1.5 ptos.) Implementa la subrutina `turn_attack` según lo especificado en Turno Ataque.

Solución:

```
turn_attack:                    # a0: direccion del primer pokemon. a1: direccion del segundo
    pokemon.
        addi sp, sp, -12
        sw ra, 0(sp)
        sw a0, 4(sp)
        sw a1, 8(sp)

        lw t0, 44(a0)
        lw t1, 44(a1)

        blt t0, t1, pokemon2_parte

        pokemon1_parte:
                jal ra, attack
                lw a0, 8(sp)
                lw a1, 4(sp)
                lw t0, 20(a0)
                beq t0, zero, retornar_turno_ataque
                jal ra, attack
                beq zero, zero, retornar_turno_ataque

        pokemon2_parte:
                lw a0, 8(sp)
                lw a1, 4(sp)
                jal ra, attack
                lw a0, 4(sp)
                lw a1, 8(sp)
                lw t0, 20(a0)
                beq t0, zero, retornar_turno_ataque
                jal ra, attack
```

```
        retornar_turno_ataque:
        lw ra, 0(sp)
        lw a0, 4(sp)
        lw a1, 8(sp)
        addi sp, sp, 12
        jalr zero, 0(ra)
```

1. (0.25 puntos) El *output* del test `test_06_turn_attack.asm` es correcto.
2. (0.25 puntos) El *output* del test `test_07_turn_attack.asm` es correcto.
3. (0.25 puntos) El *output* del test `test_08_turn_attack.asm` es correcto.
4. (0.25 puntos) El *output* del test `test_09_turn_attack.asm` es correcto.
5. (0.5 puntos) El *output* del test `test_10_turn_attack_sp.asm` es correcto.

(c) (1.5 ptos.) Implementa la subrutina `evolve` según lo especificado en Evolución.

Solución:

```
evolve:                                 # a0: direccion del pokemon a evolucionar.
        addi sp, sp, -8
        sw ra, 0(sp)
        sw a0, 4(sp)

        lw t0, 0(a0)    # ID
        lw t1, 8(a0)    # Nivel

        la t2, evolutions
        loop_evolve:
        lw t3, 0(t2)
        bgt t3, t0, retornar_evolve
        beq t0, t3, evolucionar
        addi t2, t2, 12
        beq zero, zero, loop_evolve

        evolucionar:
        lw t3, 4(t2) # Next ID
        lw t4, 8(t2) # Lvl
        blt t1, t4, retornar_evolve
        sw t3, 0(a0)
        jal ra, _print_evolve_text

        retornar_evolve:
        lw ra, 0(sp)
        lw a0, 4(sp)
        addi sp, sp, 8
        jalr zero, 0(ra)
```

1. (0.25 puntos) El *output* del test `test_11_experience.asm` es correcto.
2. (0.25 puntos) El *output* del test `test_12_experience.asm` es correcto.
3. (0.25 puntos) El *output* del test `test_13_experience.asm` es correcto.
4. (0.25 puntos) El *output* del test `test_14_experience.asm` es correcto.

5. (0.5 puntos) El *output* del test `test_15_experience_sp.asm` es corecto.

(d) (1.5 ptos.) Implementa la subrutina `experience` según lo especificado en Experiencia.

Solución:

```
experience:                             # a0: direccion del pokemon ganador de experiencia.
        lw t0, 8(a0)    # Nivel del pokemon a0
        lw t1, 28(a0)   # Ataque del pokemon a0
        lw t2, 36(a0)   # Ataque especial del pokemon a0
        add t0, t0, t1
        add t0, t0, t2
        lw t1, 12(a0)   # Experiencia del pokemon a0
        add t0, t0, t1  # Experiencia del actual pokemon a0
        sw t0, 12(a0)   # Experiencia del pokemon a0
        addi t1, zero, 100
        blt t0, t1, seguir_experience
        addi t0, t0, -100
        sw t0, 12(a0)   # Experiencia del pokemon a0
        addi sp, sp, -4
        sw ra, 0(sp)
        jal ra, _level_up
        lw ra, 0(sp)
        addi sp, sp, 4
        seguir_experience:
        jalr zero, 0(ra)
```

1. (0.25 puntos) El *output* del test `test_16_evolve.asm` es corecto.
2. (0.25 puntos) El *output* del test `test_17_evolve.asm` es corecto.
3. (0.25 puntos) El *output* del test `test_18_evolve.asm` es corecto.
4. (0.25 puntos) El *output* del test `test_19_evolve.asm` es corecto.
5. (0.5 puntos) El *output* del test `test_20_evolve_sp.asm` es corecto.

(e) (1 pto.) **Bonus:** Implementa la subrutina `poison_damage` según lo especificado en Estado: Envenenado.

Solución:

```
poison_damage:                  # a0: direccion del pokemon a verificar envenenamiento.
        addi sp, sp, -8
        sw ra, 0(sp)
        sw a0, 4(sp)

        # Verificar si esta envenenado
        lw t0, 16(a0)
        addi t1, zero, 1
        bne t0, t1, retornar_poison

        # Traer HP y restar 2
        lw t0, 20(a0)
        addi t0, t0, -2
```

```
        bgt t0, zero, seguir_poison
        # Se murio :(
        sw zero, 20(a0)
        jal ra, _print_poison_text
        lw a0, 4(sp)
        jal ra, _faint
        beq zero, zero, retornar_poison

        seguir_poison:
        sw t0, 20(a0)
        jal ra, _print_poison_text
        retornar_poison:
        lw ra, 0(sp)
        addi sp, sp, 8
        jalr zero, 0(ra)
```

1. (0.2 puntos) El *output* del test `test_21_poison.asm` es corecto.
2. (0.2 puntos) El *output* del test `test_22_poison.asm` es corecto.
3. (0.2 puntos) El *output* del test `test_23_poison.asm` es corecto.
4. (0.2 puntos) El *output* del test `test_24_poison.asm` es corecto.
5. (0.2 puntos) El *output* del test `test_25_poison_sp.asm` es corecto.

(f) (1 pto.) **Bonus:** Implementa la subrutina `wake_up` según lo especificado en Estado: Dormido.

Solución:

```
wake_up:                                # a0: direccion del pokemon a despertar.
        addi sp, sp, -8
        sw ra, 0(sp)
        sw a0, 4(sp)
        lw t0, 16(a0)

        addi t1, zero, 2
        bne t0, t1, retornar_wake_up

        addi a0, zero, 10
        jal ra, _rand_number
        add t0, zero, zero
        addi t1, zero, 5
        bge a0, t1, retornar_wake_up
        lw a0, 4(sp)
        add t0, zero, zero
        sw t0, 16(a0)
        jal ra, _print_wake_up_text
        retornar_wake_up:
        lw ra, 0(sp)
        addi sp, sp, 8
        jalr zero, 0(ra)
```

1. (0.2 puntos) El *output* del test `test_26_sleep.asm` es corecto.
2. (0.2 puntos) El *output* del test `test_27_sleep.asm` es corecto.
3. (0.2 puntos) El *output* del test `test_28_sleep.asm` es corecto.

4. (0.2 puntos) El *output* del test `test_29_sleep.asm` es corecto.

5. (0.2 puntos) El *output* del test `test_30_sleep_sp.asm` es corecto.