Parcial Arquitectura RISC-V

Ejercicio 1

El objetivo del ejercicio consiste en recibir un array de números guardados en half-word (el cual accedemos a partir de la etiqueta *arr*) e ir sumando los elementos tal que se cumpla que:

- Se enmascare únicamente los bits que se encuentran en posiciones pares con el número 0x5555 (puesto que trabajamos en half-word).

Paso 1: **Main**

Inicializo mi programa utilizando los registros a0 para llevar tracking de la suma, a1 con la dirección donde tengo cargado el array *data*, a2 con la cantidad de elementos que tendré en el array (el cual me sirve luego de iterador) y a3 hardcodeando la constante utilizada en el AND.

Llamo luego a suma_arreglo, mi función que se encarga de realizar el ejercicio.

```
# inicializar programa
main:
    li a0, 0
    la a1, arr
    li a2, 12
    li a3, 0x5555

    call suma_arreglo
```

Paso 2: **Suma_arreglo y loop**

En mi primer etiqueta, el objetivo es guardarnos en otros 4 registros los valores que habíamos cargado anteriormente en las posiciones [a0, ... , a3] para luego poder manipularlos y hacer la ejecución del loop.

En el cuerpo del loop, la lógica es la siguiente:

- Cargo en t_0 el valor almacenado en S_1 con el offset en 0.
- Realizo el *AND* entre el número guardado y la constante 0x5555.
- Acumulo en mi registro S_0 la suma de los elementos.
- Me muevo 2 bytes para recorrer mi arreglo puesto que estamos trabajando en *half-word*.

- Decremento la cuenta de mi iterador/longitud en 1.
- Repito el loop.

Aclaración: la etiqueta beqz por cada iteración está checkeando si mi longitud es 0 (es decir, no tengo más elementos para sumar), en el caso que lo fuera, salta a la etiqueta return.

```
suma_arreglo:
    mv s0, a0 # se encargará de llevar cuenta de la suma.
    mv s1, a1 # guardo i-esimo elemento del array
    mv s2, a2 # longitud del arreglo.
    mv s3, a3 # guardo la constante para hacer el AND


loop:
    beqz s2, return

    lh t0, 0(s1)
    and t0, t0, s3 # realizo el AND con mi constante y valor del elemento del arreglo

    add s0, t0, s0

    addi s1, s1, 2
    addi s2, s2, -1 # decremento la cuenta de n en 1


    j loop ## iteramos n veces
```

Paso 3: **Finalización**

Habiendo evaluado que S_2 es cero, la etiqueta return nos mueve el contenido que teníamos en S_0 a la dirección de retorno a0 (por convención) y luego llama a la etiqueta halt para aguardar instrucciones (y en definitiva cortar el programa).

```
return:
    mv a0, s0
    j halt

halt:
    j halt
```

Ejercicio 2

El objetivo de este ejercicio consiste en tomar dos arreglos *src* y *dst* cuya longitud es la misma, realizar un OR sobre ambos números y guardar el valor resultante en la i-esima posición de

dst.

Paso 1: **Main**
Al igual que el ejercicio 1, cargamos los valores que utilizaremos en las posiciones a0 hasta a2. Puesto que la convención dice que a0 y a1 son direcciones de store y retorno, guardé mi dirección a *dst* en a1.

```
main:
    # Cargamos en registros de argumentos para funciones, source y destino.
    la a0, src
    la a1, dst
    li a2, 12 ## longitud array
    call array_or
```

Paso 2: **Array_or y Loop**

```
array_or:
    mv s0, a0 # Dirección source a s0.
    mv s1, a1 # Dirección destino a s1.
    mv s2, a2 # me guardo la longitud para luego ir restando (funcionaria como mi "iterador").


loop:
    beqz s2, return  # Cuando terminamos nuestras iteraciones, volvemos a la etiqueta return

    mv t0, zero
    lw a3, 0(s0)
    lw a4, 0(s1)

    or t0, a3, a4 # Hago el "or" entre los valores de mis dos registros

    sw t0, 0(s1) # Guardo en el registro s1 el valor de OR entre a0 y a1 (que esta en t0)

    addi s0, s0, 4 # sumo 4 bytes al valor almacenado en s0 (src)
    addi s1, s1, 4 # sumo 4 bytes al valor almacenado en s1 (dst)
    addi s2, s2, -1 # decremento la cuenta de n en 1

    j loop ## iteramos
```

Misma idea, cargamos desde S_0 hasta S_2 con las respectivas posiciones de A y entramos al loop.

Secuencia del loop:

- Movemos temporalmente a t_0 la constante cero (mi valor resultante)
- Cargamos en a3 y a4 los inmediatos S_0 y S_1 (nuestros i-ésimos elementos de ambos arreglos).
- Realizamos el OR entre ambos inmediatos y los cargamos en t_0 para luego guardarlos en el registro S_1.

- Realizo un "offset" a la posición de memoria de 4 bytes tanto en *src* como en *dst* y vuelvo a realizar el loop la cantidad de elementos que tenga en el array.
- Siempre checkeo en la primer linea que mi longitud no sea igual a 0, si lo fuera salto a la etiqueta return.

Paso 3: Finalización

Puesto que nos interesa devolver la posición del array modificado (en nuestro caso *dst*), la instrucción return mueve al registro *ra* (return address) la posición donde se encuentra *dst* (guardada en a1) y salta a la etiqueta halt donde finaliza mi ejecución.

```
return:
    mv ra, a1
    j halt

halt:
    j halt
```