Ejercicio 1

Escribir un programa que recorra un arreglo de 12 elementos de 32 bits y acá la disyunción de aquellos que estén en posiciones impares.

```c
#include <stdio.h>
#include <stdint.h>
#define N 12

int main(){

    int32_t arr[N] = {
        0xFFFFFFFF, 0x95555555, 0xF4444444, 0xF1111111,
        0xFFFFFF00, 0xF5005555, 0x95444444, 0xF1113311,
        0xFF00FFFF, 0xf5550055, 0xA4444433, 0xA1551111
    };

    int32_t suma = 0x00000000;
    int32_t i = 0;
    int32_t n = 12;
    for(i = 0; i < n; i++) {
        if((i & 0x1) == 0x1 ) {
            suma |= arr[i];
        }
    }
    printf("Suma %08x", suma);

    return 0
}
```

Primer acercamiento, sin escribir código

En este caso, lo que se puede hacer es que dado un array de 12 elementos, recorra las posiciones impares (sabiendo que es de 12 elementos, sabemos que podemos indexar en 1, 3, 5, 7, 9, 11). Para generalizarlo y no tener valores *hardcodeados*, se puede optar por empezar el valor que itera en 1, e ir sumando de a 2 siempre que `i < 11`

Una vez que itero sobre esos valores, en cada iteración puedo hacer un `or` sobre la suma y el valor, y guardarlo en el mismo lugar que la suma.

Una vez terminado, debería poder hacer la *syscall* para hacer un write en `STDOUT`.

Solución

Acá voy a poner código sin comentar, en el archivo ejercicios_1_riscv.asm está comentado!

```
.text:
main:
    la a0, arr
    lw a1, longitud
    li a2, 11
    li t4, 0

loop:
    blez a2, exit
    slli t0, a2, 2
    add t0, a0, t0
    lw t0, 0(t0)
    or t4, t4, t0
    addi a2, a2, -2
    j loop

exit:
    mv a0, t4
    li a7, 34
    ecall

    li a0, 0
    li a7, 93
    ecall

.data:
    arr:
        .word 0xffffffff, 0x95555555, 0xf4444444, 0xf1111111
        .word 0xffffff00, 0xf5005555, 0x95444444, 0xf1113311
        .word 0xff00ffff, 0xf5550055, 0xa4444433, 0xa1551111
    longitud:
        .word 12
```

Vamos a explicar cada etiqueta por separado (excepto `exit` y las de `.data`)

Main

Primero cargamos la dirección del array en `a0`, la longitud del array en `a1`, el valor inicial por el que vamos a empezar a iterar en `a2` (esto podría ser `longitud - 1`, para una posible generalización del problema), y el valor que usamos como acumulador del resultado en `t4`.

Loop

`blez a2, exit` - si fuésemos a acceder a una posición menor a 0, salimos (esta es la condición de iteración, la guarda)

`slli t0, a2, 2` - esto es un poquito de "magia" a mi gusto, pero fue lo primero que se me ocurrió para resolverlo y funciona. Lo que hago es guardar en `t0` el offset que quiero usar para acceder al array en las proximas lineas. (Es como multiplicar `a2` por 4, los 4 bytes que ocupa cada elemento del vector)

`add t0, a0, t0` - le agrego el valor del iterador al offset

`lw t0, 0(t0)` - accedo al vector y me traigo ese valor a t0

`or t4, t4, t0` - hago el `or` sobre el resultado y lo que me traje, y lo sobreescribo en el resultado

`addi a2, a2, -2` - me muevo 2 para atras en el vector (11 -> 9 -> ... -> 3 -> 1)

`j loop` - vuelvo arriba para seguir iterando

Una vez que la condición se cumple (`blez a2, exit`) voy a `exit`, donde *printeo* el resultado y salgo.

Resultado

La ejecución del programa en C dio como resultado: `0xF5557755` La ejecución del programa en assembler dio como resultado: `0xF5557755`

Ejercicio 2

Escribir un programa que tome los números del arreglo fuente, aplica un or sobre éstos y el valor correspondiente del destino y lo almacena en destino.

```c
#include <stdio.h>
#include <stdint.h>
#define N 12

int main(){

    int32_t src[N] = {
        0xFFFFFFFF, 0x95555555, 0xF4444444, 0xF1111111,
        0xFFFFFF00, 0xF5005555, 0x95444444, 0xF1113311,
        0xFF00FFFF, 0xf5550055, 0xA4444433, 0xA1551111
    };

    int32_t dst[N] = {
        0xF5005555, 0x95444444, 0xF1113311, 0xFFFFFF00,
        0xF1111111, 0xFFFFFFFF, 0x95555555, 0xF4444444,
        0xA1551111, 0xFF00FFFF, 0xf5550055, 0xA4444433
    };

    int32_t i = 0;
```

```
    int32_t n = 12;
    for(i = 0; i < n; i++) {
        dst[i] |= src[i];
        printf(" %08x", dst[i]);
    }

    return 0;
}
```

Primer acercamiento, sin escribir código

En este caso, dado que hay 2 arrays de 12 elementos, puedo iterarlos a la vez, leer sus valores, hacer el `or`, y guardarlo en el mismo lugar que leí de `dst`. Puedo mover el puntero con +4 para moverme un elemento para adelante. Puedo usar la longitud como condición de terminación.

Solución

Acá voy a poner código sin comentar, en el archivo ejercicios_2_riscv.asm está comentado!

```
.text:
main:
    la a0, src
    la a1, dst
    lw a2, longitud

loop:
    blez a2, imprimir
    lw t0, 0(a0)
    lw t1, 0(a1)
    or t2, t0, t1
    sw t2, 0(a1)
    addi a0, a0, 4
    addi a1, a1, 4
    addi a2, a2, -1
    j loop

imprimir:
    la t2, dst
    li t3, 12

loop_imprimir:
    beqz t3, exit
    addi t3, t3, -1
    lw t4, 0(t2)
```

```
    mv a0, t4
    li a7, 34
    ecall
    addi t2, t2, 4
    j loop_imprimir

exit:
    li a0, 0
    li a7, 93
    ecall

.data:
    src:
        .word 0xffffffff, 0x95555555, 0xf4444444, 0xf1111111
        .word 0xffffff00, 0xf5005555, 0x95444444, 0xf1113311
        .word 0xff00ffff, 0xf5550055, 0xa4444433, 0xa1551111

    dst:
        .word 0xf5005555, 0x95444444, 0xf1113311, 0xffffff00
        .word 0xf1111111, 0xffffffff, 0x95555555, 0xf4444444
        .word 0xa1551111, 0xff00ffff, 0xf5550055, 0xa4444433

    longitud:
        .word 12
```

Vamos a explicar cada etiqueta por separado (excepto `exit`, `imprimir`, `loop_imprimir`, y las de `.data`)

Main

Ponemos en `a0`el address del array `src`, en `a1` el address del array `dst`, y en `a2` la cantidad de iteraciones/la longitud de los arrays

Loop

`blez a2, imprimir` - primero que todo, si ya recorrimos todo el array voy para `imprimir`

`lw t0, 0(a0)` - me traigo lo que está en el array `src`en la posición actual

`lw t1, 0(a1)` - me traigo lo que está en el array `dst`en la posición actual

`or t2, t0, t1` - hago `or` entre los dos valores que me traje y lo guardo en `t2`

`sw t2, 0(a1)` - guardo el valor de `t2` en el array `dst` en la posición donde tengo el puntero (es la misma que leí)

`addi a0, a0, 4` - aumento el puntero de `src` para que apunte al siguiente

`addi a1, a1, 4` - aumento el puntero de `dst` para que apunte al siguiente

`j loop` - vuelvo arriba para seguir iterando

Resultado

La ejecución del programa en C dio como resultado:

```
[0xffffffff, 0x95555555, 0xf5557755, 0xffffff11, 0xffffff11,
0xffffffff, 0x95555555, 0xf5557755, 0xff55ffff, 0xff55ffff,
0xf5554477, 0xa5555533]
```

La ejecución del programa en assembler dio como resultado:

```
[0xffffffff, 0x95555555, 0xf5557755, 0xffffff11, 0xffffff11,
0xffffffff, 0x95555555, 0xf5557755, 0xff55ffff, 0xff55ffff,
0xf5554477, 0xa5555533]
```