Organización del Computador

Trabajo Práctico 2° parte

Nombre y Apellido: Aldana Leonor Allones Carril

Padrón: 108232

1)

```
        .begin
        .org 2048
        ld %r14, %r1   !Cargar el tope de la pila en %r1
        add %r14, 4, %r14      !Acomodar la pila
        ld %r14, %r2   !Cargar el tope de la pila en %r2
        st %r15, %r14 !Guardar %r15 en la pila
        add %r14, -4, %r14     !Apuntar al tope de la pila
        st %r2, %r14   !Guardar %r2 en la pila
        add %r14, -4, %r14     !Apuntar al tope de la pila
        st %r1, %r14   !Guardar %r1 en la pila
        call prom      !Llamar a la subrutina que calcula el promedio
        ld %r14, %r3   !Cargar el tope de la pila en %r3
        add %r14, 4, %r14      !Acomodar la pila
        ld %r14, %r15 !Cargar el tope de la pila en %r15
        add %r14, 4, %r14      !Acomodar la pila
        st %r3, [z]    !Guardar el resultado en z
        jmpl %r15 + 4, %r0     !Volver al proceso invocante


prom:   ld %r14, %r8   !Cargar el tope de la pila en %r8
        add %r14, 4, %r14      !Acomodar la pila
        ld %r14, %r9   !Cargar el tope de la pila en %r9
        addcc %r8, %r9, %r10           !Sumar ambos números y guardar el resultado en %r3
        sra %r10, 1, %r10      !Dividir %r3 por 2 y guardar el resultado en %r3
        st %r10, %r14 !Guardar %r10 en la pila
        jmpl %r15 + 4, %r0     !Volver al programa principal
z: 0
        .end
```

Tal como ajustaste el stack pointer tras la lectura el primer sumando, habrá que ajustarlo inmediatamente después de la lectura del segundo sumando.

Acá nuevamente el stack pointer quedaría desactualizado. Siempre pensar que las dos operaciones del push y las dos operaciones del pop son una ***transacción***. Así evitamos dejar rastros en el stack.

Entiendo que %r10, no %r3

2)

ORCC	JMPL
IF R[IR[13]] THEN GOTO YYYY	IF R[IR[13]] THEN GOTO XXXX
R[rd] ← ORCC(R[rs1], R[rs2]); GOTO 2047;	R[pc] ← ADD(R[rs1], R[rs2]); GOTO 0;
YYYY: R[temp0] ← SIMM13(R[ir]);	XXXX: R[temp0] ← SEXT13(R[ir]);

No, en realidad ORCC es una instrucción lógica y JMPL es una instrucción de control de flujo de programa.

R[rd] ← ORCC(R[rs1], R[temp0]); GOTO 2047;	R[pc] ← ADD(R[rs1], R[temp0]); GOTO 0;

Ambas son instrucciones aritméticas, por lo tanto, hay dos formatos posibles para las instrucciones:

a. Los 13 LSB son ocupados por un valor inmediato, es decir, una constante
b. Los 6 LSB son ocupados por el número de un registro de propósito general (rs1)

La manera que tenemos de saber cuál es, es mirando el bit 13 del registro de instrucción.

Si se trata de la primera, IF R[IR[13]] es True y salta a la dirección YYYY o XXXX donde debe extender la constante de 13 bits a 32 bits:

a. En orcc, SIMM13(R[ir]) agrega ceros en los 19 MSB de la constante
b. En jmpl, SEXT13(R[ir]) agrega unos o ceros dependiendo si la constante es un número negativo o positivo, respectivamente

De esta manera, la constante se extiende al ancho de palabra sin alterar su valor, por lo tanto, sin alterar el resultado de las operaciones.

La constante de 32 bits se guarda en un registro temporal, y es este el que se utiliza en la instrucción siguiente junto con el registro de propósito general para realizar la operación.

Si se trata de la segunda, IF R[IR[13]] es False y se realiza la operación de la segunda fila utilizando los dos registros como operandos.

En ORCC: GOTO 2047; se dirige a la dirección que incrementa el PC.

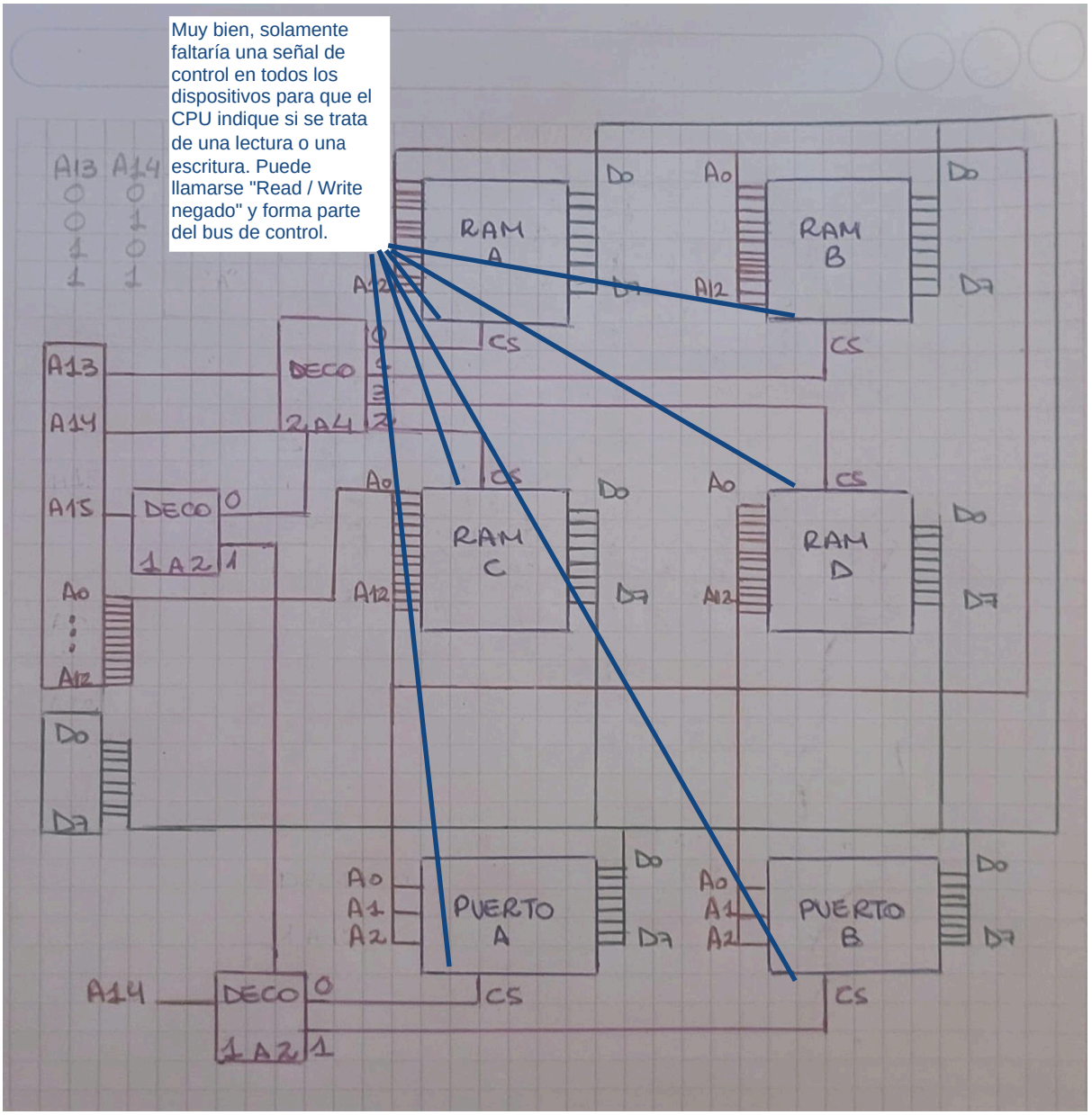
En JMPL: GOTO 0; se dirige a la dirección donde vuelve a comenzar el ciclo. Esta diferencia se debe a que el PC ya se actualizó.

Perfecto

3.

2**10 - 1K
2**11 - 2K
2**12 - 4K
2**13 - 8K

Para direccionar 32KB de RAM se necesitan 4 chips de RAM de 8K.



	Primera dirección	Última dirección
RAM A	0x0000	0x1FFF
RAM B	0x4000	0x5FFF
RAM C	0x2000	0x3FFF
RAM D	0x6000	0x7FFF
PUERTO A	10xx xxxx xxxx x000	10xx xxxx xxxx x111
PUERTO B	11xx xxxx xxxx x000	11xx xxxx xxxx x111

RAM A: A15 = 0, A14 = 0 y A13 = 0

Muy bien.(Los puertos, por todas las "xxxx"
quedan mapeados como memoria fantasma)

- Primera dirección: 0000 0000 0000 0000 = 0x0000

Se verifica que las direcciones de la RAM empiezan en la dirección $0.

- Última dirección: 0001 1111 1111 1111 = 0x1FFF

RAM B: A15 = 0, A14 = 1 y A13 = 0

- Primera dirección: 0100 0000 0000 0000 = 0x4000
- Última dirección: 0101 1111 1111 1111 = 0x5FFF

RAM C: A15 = 0, A14 = 0 y A13 = 1

- Primera dirección: 0010 0000 0000 0000 = 0x2000
- Última dirección: 0011 1111 1111 1111 = 0x3FFF

RAM D: A15 = 0, A14 = 1 y A13 = 1

- Primera dirección: 0110 0000 0000 0000 = 0X6000
- Última dirección: 0111 1111 1111 1111 = 0x7FFF

Muy bien

PUERTO A: A15 = 1, A14 = 0

- Primera dirección: 10xx xxxx xxxx x000
- Última dirección: 10xx xxxx xxxx x111

PUERTO B: A15 = 1, A14 = 1

- Primera dirección: 11xx xxxx xxxx x000
- Última dirección: 11xx xxxx xxxx x111

De A13 a A3 podrían ser unos o ceros.