Marco Ferraro Rodriguez | B82957
Gabriel Revillat Zeledón | B86524
Examen 1: Arquitectura de Computadoras

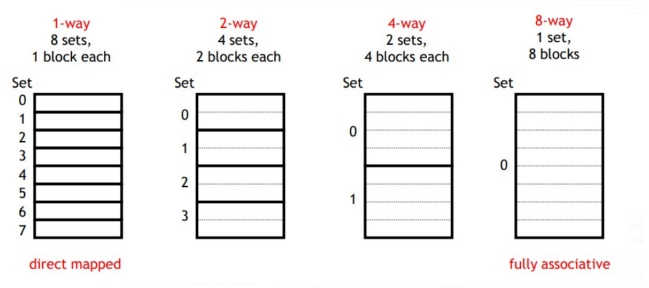
A.

A.1.

Composición de Memoria Cache:

- Componentes de las direcciones de 16 bits
  - Índice: En una dirección de 16 bits, se tienen un conjunto de bits reservados que indican en que **set** de la memoria cache se tiene que guardar los datos.
  - Etiqueta: Ya que varias direcciones de memoria pueden ser mapeadas a un mismo set, hay bits reservados que diferencian estas direcciones bajo el mismo set. El valor es conocido como **tag** o **etiqueta**
  - Offset: El offset es un valor que sirve para "marcar un iterador"de que byte específico acceder. Esto se debe que aunque la unidad más pequeña equivale a un word, puede que se desee acceder a un byte específico de esta unidad de almacenamiento.
- Condiciones para un "miss"

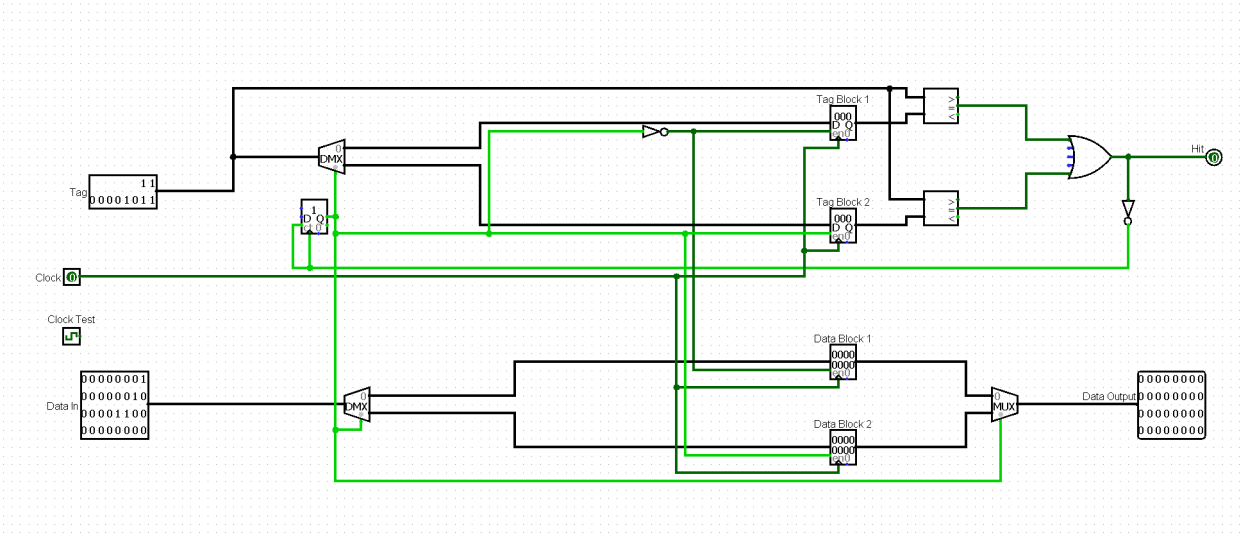
  Un miss en la cache puede ocurrir por varias condiciones. Una es que la cache **no esté inicializada**, por lo que al cargar una dirección de memoria va a dar miss, ya que la cache está sin valores. Otra condición es cuando la cache esta cargada con direcciones de memoria, pero la que se quiere referenciar no está en la memoria cache. Se identifica que no está por medio del tag.
- Componentes de cada linea de cache
- Valid Bit: El bit de validez es para indicar si la cache ya ha sido inicializada o no. Por lo tanto, cuando la cache esté vacia, el bit de validez de esa linea va a estar en 0.
- Dirty Bit: El dirty bit nos ayuda a identificar si el valor de la dirección de memoria que está en cache, está actualizado con respecto al valor que esta en memoria principal.
- Bloque y Set El concepto de bloque y set tiende a confundirse, por eso veamos está imagen.



En 1 nivel de memoria cache, el bloque de cache hace referencia a la unidad de almacenamiento. Los datos de un bloque de datos de la memoria principal van a estar almacenados en 1 bloque de la cache. Los set corresponden a como se asocian estós bloques dentro de la cache.

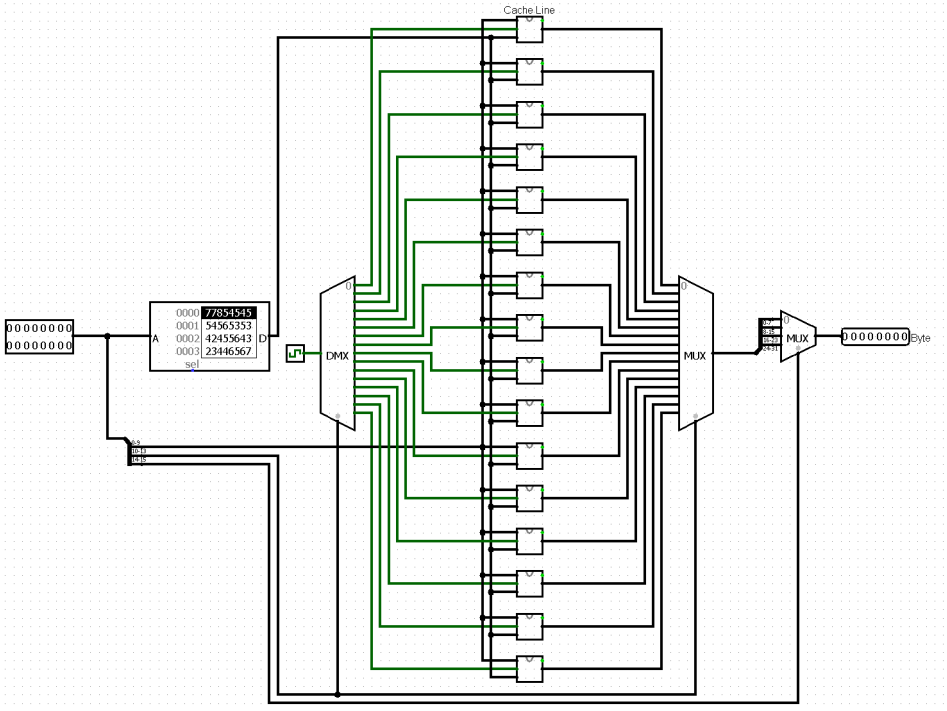
A.2. Circuito Linea Cache Logisim

Para esta seccíon se nos pide un circuito en logisim que simule una linea o conjunto de cache.



Como se aprecia en la imagen, la lógica de este circuito se basa en un modelo 2-way set asociative. Por lo tanto, se manejan 2 bloques para el tag, y 2 para los datos. Si ocurre un miss, se escoje 1 de los 2 bloques para actualizar la información. Si ocurre un hit, se marca prende el pin respectivo.

A.3. Circuito principal



Con las herramientas que nos provee logisim, podemos montar un circuito principal, que incluya el subcircuito anterior. Podemos ver que este circuito principal incluye un reloj; desde acá se controla todo el flujo del circuito. Además incluimos una memoria ROM para mantener las instrucciones (por ser logisim y por el alcance de este examen, decidimos incluir memoria ROM con valores ya definidos en vez de una memoria principal, ya que no se va a usar todas sus funcionalidades.)

B.

Recordemos:

Según la teoría, la propagación por copia es una técnica de optimización que se maneja desde el código ensamblador. La intención de esto es eliminar el código dependiente, por lo tanto, si una variable ya tiene un valor definido anteriormente, se puede copiar este valor, para reducir el número de variables en una expresión.

a = b + c // La propagación por copia no le hace nada a esta instrucción.
b = a + c = (b + c) + c // La transformación aumenta el trabajo computacional en una suma.
d = a – b = (b + c) - ((b + c) + c) = -c // La transformación aumenta el trabajo en tres sumas. Pero como se puede simplificar, reduce el trabajo computacional de una resta a una expresión negativa.

C.

Promedio de los datos de los programas *libquantum* y *mcf*

Instrucción	*libquantum*	*mcf*	Promedio
ALU Operations	47	28	37,5
Loads	16	35	25,5
Stores	6	11	8,5
Branches	29	24	26.5
Jumps	0	1	0.5

Consideramos que el 60 % de las ramas son tomadas.
CPI Efectivo = Porcentaje de instrucción * Ciclos de reloj
CPI Efectivo = (0.375 * 1) + (0.255 * 5) + (0.085 * 3) + (0.265 * (0.6 * 5 + (1 - 0.6) * 3)) + (0.005 * 3) = 3.033

D.

	Ciclo																	
Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
load x1, 0(x2)	IF	ID	EX	MEM	WB													
addi x1, x1, 1		IF	stall	stall	ID	EX	MEM	WB										
sd x1, 0, (x2)					IF	stall	stall	ID	EX	MEM	WB							
addi x2, x2, 4								IF	ID	EX	MEM	WB						
sub x4, x3, x2									IF	stall	stall	ID	EX	MEM	WB			
bnez x4, Loop												IF	stall	stall	ID	EX	MEM	WB