Eran tres ejercicios pero habia distintas alternativas en cada punto.

1)

Dadas dos matrices cuadradas A y B que contienen 1024 enteros cada una y cada entero es de 4 bytes.
El primer elemento de A (A[0,0]) está almacenado en la dirección física 0000 4020 H y el primer elemento de B en la dirección física 0001 0800H.

El sistema tiene una caché de 4KB para datos, organizada en mapeo directo con líneas de 16 bytes y política de escritura *write through - write allocate*.

a. ¿Cuántas líneas hay en la caché?
b. ¿En qué campos y de qué cantidad de bits cada uno se divide la dirección física para acceder a la caché?
c. Indique en cuántos bloques se dividen las matrices y qué set le corresponde a cada uno.

Si inicialmente todas las entradas de la caché están marcadas como inválidas y durante la ejecución del siguiente código solamente A y B se almacenan en la caché porque *i* y *j* se almacenan en registros.

```
for (i = 0; i < 32; i++) {
    for (j=0; j < 32; j++) {
        A[i,j] = i+j;
        B[j,i] = B[j,i] + A[i,j];
    }
}
```

d. ¿Cuántos fallos de lectura y de escritura ocurren? ¿Ese valor hubiera variado si la caché, manteniendo la capacidad para datos, hubiera estado organizada con un grado de asociatividad 2?
e. ¿Cuántos bytes se escribieron en memoria principal?

2)

ALTERNATIVA 1:

Suponiendo el siguiente escenario para una implementación simple del pipeline de 5 etapas:

- Predicción de *Branch* y *branch target buffer* con 20% de predicciones erróneas.
- Hay *forwarding* sólo para datos generados por operaciones aritméticas y todos los demás conflictos se resuelven con *Stall*.
- Tanto las instrucciones *branch* como *jump* actualizan el PC en la etapa *Memory*.
- De las instrucciones, 30% son *load,* 10% son *branch y* 2% son *jump.*

a. Calcule qué porcentaje debe haber de *load* seguidos de una instrucción que usa el dato para obtener un *speed up* de 4 en comparación con una implementación sin pipeline.

b. Si se mejora el hardware agregando *forwarding* también para datos leídos de memoria y adelantando a la etapa *Decode* la resolución de los *jumps* y al *Execute* la de los *branchs*, ¿cuál es el *speed up* de esta implementación en comparación con la anterior?

ALTERNATIVA 2:

Supongamos dos microarquitecturas M1 y M2 del mismo set de instrucciones, con los siguientes CPI.

Operations	M1	M2
Load–store	1	2
ALU	2	2
Branches	3	2

Nos interesa analizar el tiempo de ejecución de dos programas A y B, que tienen la siguiente proporción de cada instrucción:

Operations	A	B
Load–store	20%	50%
ALU	50%	30%
Branches	30%	20%

a. Si la frecuencia del reloj de M1 es 2 GHz. ¿Cuál debería ser la frecuencia del reloj de M2 para que el programa A tenga el mismo tiempo de ejecución en M1 que en M2? ¿Y para que ocurra eso con el programa B? ¿A qué conclusión puede llegar en función de los resultados obtenidos?

b. Supongamos que la frecuencia del reloj de M1 y M2 es la misma, ambos programas tiene la misma cantidad de instrucciones y para realizar determinada tarea encontramos un algoritmo que requiere ejecutar A y B la misma cantidad de veces ¿qué microarquitectura va a ser más rápida?

c. ¿Cuál debería haber sido la relación entre la cantidad de ejecuciones de A y la cantidad de B para que en el inciso anterior el resultado fuera que son igual de rápidas?

3)

ALTERNATIVA 1:

Estás diseñando un sistema de memoria virtual que usa una tabla de páginas de un único nivel implementada en hardware dedicado (una memoria SRAM y la lógica asociada). Soporta direcciones virtuales de 25 bits, direcciones físicas de 22 bits y páginas de 2^{16} bytes. Cada entrada de la tabla de páginas contiene el número de *frame*, el bit de válido y el bit de *dirty*.

a. ¿Cuál es el tamaño total en bits de la tabla de páginas?
b. El equipo que diseña el sistema optativo propone reducir el tamaño de la página a 16KB, pero los integrantes del equipo que diseña el hardware (tu equipo) lo rechazan basándose en el costo del hardware adicional. Explica el porqué de tal objeción.
c. La tabla de páginas va a ser integrada en el chip del procesador, junto con la caché (que soporta solo direcciones *físicas*, no virtuales). Para un dado acceso a memoria, ¿es posible acceder el *set* correcto en la caché concurrentemente con el acceso a la tabla de páginas? Explique qué debe ocurrir para que esto sea posible.
d. Para un dado acceso a memoria, ¿es posible realizar la comparación del *tag* en la caché concurrentemente con el acceso a la tabla de páginas? Explique qué debe ocurrir para que esto sea posible.

ALTERNATIVA 2:

Supongamos que sos parte del equipo de diseño de una máquina. Ya está decidido que el sistema podrá direccionar un espacio virtual de 2^{32} bytes dividido en páginas de 4KiB. El espacio en disco no es un problema, pero el hardware limita la memoria RAM a un máximo de 8MiB.

a. ¿Cuántos bits tiene la dirección física?
b. ¿En cuántas páginas se divide el espacio virtual?
c. ¿Cuál es el máximo número de *frames* posible?
d. ¿Cuántos bits tiene el número página y el número de *frame*?
e. Alguien del equipo propone usar un esquema de mapeo directo entre páginas y *frames*. El mapeo usaría los últimos bits del número de página para determinar el número de *frame*. Sorprendentemente, a la mayoría del equipo le está gustando la idea ¿qué argumentos usarías para convencer al equipo de que esta es una mala idea y no debe prosperar?
f. Finalmente se decide usar una tabla de páginas para almacenar ese mapeo. La tabla además del número de *frame* debe almacenar 2 bits extras de estado (bit de válido y bit de *dirty*). ¿Cuál será el tamaño, en cantidad entera de bytes, de esa tabla?
g. Al calcular el tamaño de la tabla (inciso anterior) el equipo de diseño se encuentra con otro problema. ¿Cuál es ese problema? Explique la estrategia que propondrías para solucionarlo.