U.B.A. - Facultad de Ingeniería

66.20 Organización de Computadoras
Guía de ejercicios

Práctica jueves

1. Principios fundamentales

1.1.

Demostrar que

$$SU = \frac{1}{(1-f) + \frac{f}{SU_l}}$$

Donde f es la fracción de tiempo a mejorar, y SU_l es la mejora local.

1.2.

Un procesador de 300 Mhz ejecuta un programa que presenta los siguientes tipos de instrucciones:

Tipo de instrucción	Frecuencia (%)	Ciclos
Aritmético-Lógica	40	1
Carga	20	1
Almacenamiento	10	2
Saltos	20	3
Punto Flotante	10	5

a) Calcular las tasas de CPI y MIPS, para el programa completo.

b) Suponga que una optimización elimina un 30 % de las instrucciones aritmético-lógicas (o sea, 12 % del total de instrucciones A-L), 30 % de instrucciones load y 20 % de punto flotante. ¿Cuál es el speedup alcanzado?

c) Recalcular las tasas de CPI y MIPS para el programa completo. Explicar las diferencias respecto al punto (a).

1.3.

Se proponen 3 mejoras para una nueva arquitectura con los siguientes Speedups:

- Speedup1 = 30
- Speedup2 = 20
- Speedup3 = 10

Sólo una mejora es aplicable en cada momento (no se pueden solapar).

a) Si las mejoras 1 y 2 se pueden usar un 30 % del tiempo, ¿qué fracción del tiempo se debe usar la mejora 3 para lograr un speedup global de 10?

b) Asumir que la distribución del uso de las mejoras es del 30 %, 30 % y 20 para las mejoras 1, 2 y 3 respectivamente. Asumir que las 3 mejoras están en uso. ¿Qué fracción del tiempo mejorado no tiene una mejora en uso?

c) Asumir que para un benchmark la fracción del uso de las mejoras es del 15 % para 1 y 2 y del 70 % para la mejora 3. Se quiere maximizar la performance. Si sólo una mejora puede ser aplicada, ¿cuál debería ser elegida? Si 2 mejoras pueden ser aplicadas, ¿cuales deberían ser elegidas?

1.4.

Se dispone de un benchmark que contiene 195, 578 instrucciones de punto flotante.

Dicho Benchmark fue ejecutado en un procesador embebido luego de haber sido compilado con las optimizaciones activadas. El procesador embebido está basado en un procesador RISC que incluye unidades de punto flotante, pero el procesador embebido no dispone de ellas por distintas razones. El compilador permite calcular las operaciones de punto flotante mediante unidades punto flotante o mediante rutinas de software, dependiendo en las opciones utilizadas.

El benchmark se ejecutó en 1, 08 segundos en el procesador RISC, mientras que tomó 13,6 segundos en la versión embebida. Asumir que el CPI del procesador RISC es 10, mientras que el CPI del procesador embebido es 6.

a) Para ambos procesadores, ¿cuántas instrucciones fueron ejecutadas?

b) Para ambos procesadores, ¿cuál es el valor de la tasa de MIPS?

c) En promedio, ¿cuántas instrucciones enteras son necesarias para ejecutar una operación de punto flotante en software?

Responder considerando que el benchmark puede estar conformado por

- Un 100 % de instrucciones de punto flotante.
- Instrucciones de punto flotante y enteras.

1.5.

El siguiente es el algoritmo de Booth, utilizado para realizar multiplicaciones binarias de X * Y utilizando sumas, restas y desplazamientos a derecha:

1. $A = 0, Q = X, Q_{-1} = 0, M = Y$
2. For I = 1 to n do
   a) $if\ Q_0Q_{-1} = 01\ then\ A = A + M$
   b) $if\ Q_0Q_{-1} = 10\ then\ A = A - M$
   c) $Arithmetic\ Right\ Shift\ (A||Q)$

La respuesta es almacenada en la combinación de $(A||Q)$

El 5 % de todas las instrucciones de un *benchmark* son multiplicaciones binarias de 32 bits (sin ocurrir otras multiplicaciones). Asumir que cada instrucción (línea) en el algoritmo toma 1 ciclo de reloj (incluyendo el for y cada if).

Asumir que la máquina que realiza la multiplicación binaria utilizando el hardware dedicado tiene un CPI de 15.

a) ¿Cuánto más rápida debe ser la máquina con una unidad de multiplicación por hardware que otra máquina que debe realizar la multiplicación utilizando el algoritmo de Boot?

2. Arquitectura de programación

2.1.

Pasar el siguiente código C a MIPS.

```
/*
        Asumir que las seis variables (f,g,h,i,j,k) corresponden a
            seis registros
($s0,$s1,$s2,$s3,$s4,$s5), y existe una variable $t2=4.
        */

switch (k) {
  case 0:
            f = i + j; break;
  case 1:
            f = g + h; break;
  case 2:
            f = g - h; break;
  case 3:
            f = i - j; break;
}
```

Pista: usar la variable k para indexar una tabla de saltos, y luego saltar a dichos valores. Primero verificar que el valor k se corresponda con una de las opciones posibles (0 <= k <= 3). Si no, salir.

2.2.

Instruction	gap	gcc	gzip	mcf	perl	I. average
load	44.7%	35.5%	31.8%	33.2%	41.6%	37%
store	10.3%	13.2%	5.1%	4.3%	16.2%	10%
add	7.7%	11.2%	16.8%	7.2%	5.5%	10%
sub	1.7%	2.2%	5.1%	3.7%	2.5%	3%
mul	1.4%	0.1%				0%
compare	2.8%	6.1%	6.6%	6.3%	3.8%	5%
cond branch	9.3%	12.1%	11.0%	17.5%	10.9%	12%
cond move	0.4%	0.6%	1.1%	0.1%	1.9%	1%
jump	0.8%	0.7%	0.8%	0.7%	1.7%	1%
call	1.6%	0.6%	0.4%	3.2%	1.1%	1%
return	1.6%	0.6%	0.4%	3.2%	1.1%	1%
shift	3.8%	1.1%	2.1%	1.1%	0.5%	2%
and	4.3%	4.6%	9.4%	0.2%	1.2%	4%
or	7.9%	8.5%	4.8%	17.6%	8.7%	9%
xor	1.8%	2.1%	4.4%	1.5%	2.8%	3%
other logical	0.1%	0.4%	0.1%	0.1%	0.3%	0%
load FP						0%
store FP						0%
add FP						0%
sub FP						0%
mul FP						0%
div FP						0%
mov reg-reg FP						0%
compare FP						0%
cond mov FP						0%
other FP						0%

Considerar el agregado de un nuevo modo de acceso a MIPS. El nuevo modo suma dos registros y un valor de offset de 11 bits con signo para obtener la dirección efectiva. Utilizar el porcentaje de instrucciones indicado en la tabla anterior. El compilador pasa de las instrucciones:

```
add R1, R1, R2
lw  Rd, 100(R1) #(o store)
```

A utilizar:

```
lw Rd, 100(R1,R2)
```

a) Asumir que el nuevo modo de acceso es usado por el 10 % de los loads y stores. ¿Cual es el porcentaje de Ic nuevo comparado con la tasa original?

b) Si el nuevo modo de direccionamiento aumenta en 5 % el tiempo de clock, ¿Cuál computadora será más rápida y por cuanto?

2.3.

Codificar en Assembly MIPS y diagramar el stack de las siguientes funciones.

```
void proc(int i){
  int j;
  j = i+20;
}
int main(int argc, char** argv){
    int i=10;
    proc(i);
    return 0;
}
```

2.4.

Codificar en Assembly MIPS y diagramar el stack de la siguiente función.

```
unsigned int
factorial (unsigned int n)
{
    if (n < 2)
        return 1;
    else
        return n*factorial(n-1);
}
```

2.5.

Codificar en C y Assembly MIPS una función que reciba calcule la longitud de un C-string

```
size_t strlen(const char *s)
```

3. Jerarquia de memoria

3.1.

Sea una computadora MIPS32, con un CPI ex de 2 y una frecuencia de clock de 1GHz, con una memoria que tiene 100ns de latencia y 200MHz de ancho de banda, en la que ejecutamos el siguiente programa:

```
$L9:
    addu    $v0,$v0,$a1
    sw      $v1,0($v0)
    addu    $v1,$v1,1
    sltu    $v0,$v1,$a0
    sll     $v0,$v1,2
    bne     $v0,$zero,$L9
```

Si el registro `$a0` contiene el valor 200, el registro a1 contiene el valor `0xcafebeb0`, y el programa está almacenado a partir de `0x0000f000`

a) ¿Cuánto tarda en ejecutarse el programa?

b) ¿Qué proporción de este tiempo ocupan los ciclos de stall?

c) Si agregamos un caché split LRU de mapeo directo y bloques de 16 bytes, de 32KB de tamaño tanto para L1I como para L1D, con un T_{hit} que no retrasa a la CPU, ¿cuánto tardaría en ejecutarse el programa si L1D es WB/WA?

d) ¿Cuáles serían los conjuntos ocupados en las dos caches?

e) ¿Cuánto tardaría si L1D es $WT/\neg WA$?

3.2.

Se dispone de una computadora MIPS32 con 32MB de memoria, sin niveles de cache e inicialmente sin soporte para memoria virtual. La memoria tiene un tiempo de acceso promedio de 100 ciclos. En esta computadora se ejecuta la siguiente función:

```
void mat_transpose(int **mat, int **mat_trans, size_t n) {
    register unsigned int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j <= i; j++) {
            mat_trans[j][i] = mat[i][j];
            mat_trans[i][j] = mat[j][i];

        }
    }
}
```

La función `mat_transpose()` recibe una matriz de enteros como primer argumento, y escribe en la dirección indicada por `mat_trans` la matriz transpuesta. Las matrices son cuadradas, de $n \times n$ enteros.

a) Si la computadora no dispone de soporte para memoria virtual, ¿cuál es el máximo tamaño que puede tener cada matriz?

b) A partir de este punto se decide brindar soporte para memoria virtual. En un principio, el único parámetro definido es que el tamaño de página es de 4 KB. Si se utilizan matrices cuadradas de 32×32, indicar cuanta memoria se necesita para soportar el sistema de traducción para cada caso, siempre considerando que cada entrada de la tabla ocupa 4 bytes.

i) Si se utiliza una tabla de páginas lineal.

ii) Si se utiliza una tabla de páginas jerárquica de dos niveles, de manera que cada tabla ocupa una página de memoria física.

iii) Si se utiliza una tabla de páginas invertida (sin HAT).

c) Finalmente, se decide utilizar una tabla jerárquica de dos niveles. Indicar el tiempo promedio de acceso a los datos de las matrices, considerando únicamente la ejecución de la función `mat_transpose`.

d) Ídem, considerando que se agrega un TLB completamente asociativo de 64 entradas.

3.3.

Se brinda a continuación una serie de referencias a memoria, dadas como direcciones de palabras: 2, 3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6 y 11. Suponer una memoria caché de correspondencia directa con 16 bloques de una palabra inicialmente vacíos. Rotular cada referencia de la lista como "acierto"(*hit*) o "desacierto" (*miss*) y mostrar el contenido final de la caché.

3.4.

Usando la serie de referencias dadas en el ejercicio anterior, mostrar los aciertos y desaciertos y el contenido final de la memoria caché para una organización de correspondencia directa con bloques de cuatro palabras y un tamaño total de 16 palabras.

3.5.

Suponer que se tienen dos computadoras idénticas A y B, salvo por su organización de memoria caché. Se pide la escritura de dos códigos de manera tal que se cumpla que el primer código se ejecute mucho más rápido en la máquina A que en la B, y que el segundo código se ejecute mucho más rápido en la máquina B que en la A.

Datos:

- Caches unificadas.
- El tiempo de escritura de una palabra de 32 bits en memoria principal es igual a 5 tiempos de acierto en memoria caché.
- Penalidad de desacierto = 10 tiempos de acierto.
- Caché A: 128 conjuntos, dos elementos por conjunto, bloque de 32 bytes, *write through* y *no-write allocate*.
- Caché B: 256 conjuntos, un elemento por conjunto, bloque de 32 bytes, *write back* y *write allocate*.

3.6.

Dado el siguiente pseudocódigo:

```
int array[10000,100000];

para cada elemento array[i][j]
    array[i][j] = array[i][j]*2;
```

1. Escribir dos programas en C que implementen este algoritmo: uno debe acceder los elementos del array recorriéndolo por filas (*row-major order*) y el otro debe accederlos recorriéndolo por columnas (*column-major order*).

2. Compare los tiempos de ejecución de los dos programas.

3. ¿Cómo explica los resultados obtenidos en base al impacto que el código ha producido en la jerarquía de memoria?

3.7.

Suponer dos memorias caches: C1 es una organización de correspondencia directa con 16 bloques de una palabra y C2 es una organización de correspondencia directa con bloques de cuatro palabras y un tamaño total de 16 palabras. Suponer una penalidad de desacierto para C1 de 8 ciclos de reloj de *bus* y una penalidad de desacierto para C2 de 11 ciclos de reloj de *bus*. Suponiendo que las caches están inicialmente vacías, encontrar una secuencia de referencias para la cual C2 tenga una tasa de desaciertos (*miss rate*) más baja pero pase más ciclos de reloj en desaciertos de caché que C1. Usar direcciones de palabras.

3.8.

Para una computadora con CPU MIPS R2000 y memoria caché de 16-KB con asociatividad de grado 4 y línea de 16 bytes:

1. Calcular el número de bits de *tag*, *índice* y *offset*.

2. Dibujar un esquema detallado de la memoria caché (numerar los conjuntos desde 0 en adelante).

3. Dar la dirección (en hexadecimal) correspondiente a cada uno de los siguientes datos y ubicarlos en el esquema del ítem b:

   a) A y B son datos de un mismo bloque de memoria que mapean en el conjunto número 7.

   b) C y D son datos de distintos bloques de memoria que mapean en el conjunto número 255.

   c) E y F son datos que pertenecen a los bloques de memoria principal siguientes a C y D, respectivamente.

   d) G es un dato separado en 1-MB de distancia respecto a B.

3.9.

Dar una secuencia de 5 direcciones de memoria distintas generadas por una CPU MIPS con memoria caché, de manera tal que se cumpla con lo siguiente:

- La segunda dirección es acierto.
- La tercera mapea en un conjunto distinto al correspondiente a la primera.
- La quinta es desacierto.

Las direcciones se deben dar en hexadecimal. La memoria caché es de 8-KB, asociativa de grado 2 y bloque de 16 bytes. No se pueden hacer hipótesis acerca del contenido previo de la caché. Justificar su respuesta.

3.10.

Para el siguiente código:

```
        add $t2, $0, $0
loop:   sll $t1, $t2, 2
        sw $t2, array($t1)
        addi $t2, $t2, 1
        slti $t3, $t2, 12
        bgtz $t3, loop
        ori $v0, $0, 10
        syscall

.data
.align 5
array:  .word 0, 1, 2, 3, 4, 5, ...
```

1. Calcular el CPI promedio.
2. Calcular el promedio de accesos a memoria por instrucción.
3. Calcular la tasa de desaciertos.

Datos: caché de 8-KB, de 2 vías asociativa por conjunto, unificada, con bloque de 32 bytes y política de reemplazo LRU. Aclare las hipótesis que utilice.

3.11.

Para una computadora con CPU MIPS R2000 y memoria caché de 512 bytes con asociatividad de grado 2 y línea de 16 bytes.

1. Calcular el número de bits de *tag*, *índice* y *offset*.
2. Dibujar un esquema detallado y completo de la memoria caché (área de datos y *tags*). Numerar los conjuntos desde 0 en adelante.
3. Ubicar en el dibujo a cada uno de los siguientes datos (bytes) dados por su dirección, y dar y ubicar en el área de *tags*, a los *tags* correspondientes.

Dato	Dirección
A	0xffffff00
B	0xffffff04
C	0xcccccc08
D	0xffffff1a
E	0xffffff70
F	0xffffffff

Justifique todas sus respuestas y muestre sus cálculos.

3.12.

Supongamos tener 3 computadoras con las siguientes configuraciones de memoria caché: (C1) DM, (C2) 2WSA y (C3) FA (i.e completamente asociativo). En todos los casos, se trata de sistemas de alojamiento unificados, en donde la capacidad es de 8 líneas de 32 bit cada una, con reemplazo LRU.

```
   # Benchmark B0
      li    t1, 100
loop: lw    t2, 1024(zero)
      subu  t1, t1, 1
      bnez  t1, loop
```

```
   # Benchmark B1
      li t1, 100
loop: lw t2, 1028(zero)
      subu t1, t1, 1
      bnez t1, loop
```

```
 # Benchmark B2
      li    t1, 100
loop: lw    t2, 1028(zero)
      lw    t2, 1032(zero)
      subu  t1, t1, 1
      bnez  t1, loop
```

```
 # Benchmark B3
      li t1, 100
loop: lw t2, 1028(zero)
      lw t2, 1032(zero)
      lw t2, 1036(zero)
      subu t1, t1, 1
      bnez t1, loop
```

```
 # Benchmark B4
      li t1, 100
loop: lw t2, 1028(zero)
      lw t2, 1032(zero)
      lw t2, 1036(zero)
      lw t2, 1040(zero)
      subu t1, t1, 1
      bnez t1, loop
```

Suponiendo que todos los *benchmarks* comienzan con la primera instrucción en la dirección de memoria nula.

1. ¿Cuál es el *benchmark* con mejor desempeño en el caché C1?
2. ¿Cuáles de 0 % 25 % 50 % 75 % 100 % representa mejor la tasa de acierto para B1+C1?
3. ¿Qué caché tiene mejor tasa de acierto con B3?
4. ¿Cuál caché, logra una tasa de acierto nula con B4? Elegir uno de C1, C2, C3, o ninguno.

3.13.

Para el siguiente sistema:

- Microprocesador: 1Ghz
- CPI (sin accesos a memoria): 0,7
- Del total de instrucciones, 20 % son lecturas, 5 % son escrituras
- Cache L1 Instrucciones:
  - Direct Mapped
  - 32KB de Tamaño
  - 32 bytes por bloque
  - Write Back.
  - 2 % de tasa de miss. Tiempo de hit despreciable.
- Cache L1 Datos:
  - Direct Mapped
  - 32KB de Tamaño
  - 32 bytes por bloque
  - Write Through
  - 5 % de tasa de miss. Tiempo de hit despreciable.
- Cache L2:
  - 512Kb de Tamaño
  - 32 bytes por bloque
  - Write Back
  - 15ns de tiempo de hit
  - 80 % de tasa de hit
  - 50 % de bloques en estado dirty
- Memoria Principal:
  - 40ns de latencia
  - Ancho de bus 64 bits

  - Clock de memoria principal 5ns
  - 200Mhz de Ancho de banda

- ¿Cuál es el tiempo promedio de acceso a memoria para leer una instrucción?
- ¿Cuál es el tiempo promedio de acceso a memoria para leer un dato?
- ¿Cuál es el tiempo promedio de acceso a memoria para escribir un dato?
- ¿Cuál es el CPI promedio, teniendo en cuenta los accesos a memoria?

3.14.

Para este problema, asumir que se tiene un procesador con un caché conectado a memoria principal vía un bus. Un caché hit toma 1 ciclo. Luego de un miss, un bloque se transfiere desde memoria principal sobre el bus. El fetch del bloque no se inicia hasta que comienza el ciclo siguiente al del miss.

Una transacción en el bus consiste de un ciclo para enviar la dirección de memoria, 4 ciclos de idle time para el acceso a memoria principal, y luego un ciclo para transferir cada word del bloque desde memoria principal hacia el caché.

Asumir que el procesador continúa la ejecución sólo después de que el último word del bloque ha sido transferido.

En otras palabras, si el tamaño de bloque es B words (32 bits/word), un caché miss costará 1+1+4+B ciclos.

El cuadro 1 muestra el miss rate promedio del caché para un caché de 1MB para varios tamaños de bloques.

B [words]	m [%]
1	3.4
4	1.1
8	0.43
16	0.28
32	0.19

Cuadro 1: Tamaño de bloque (B) vs. Miss rate (m)

1. Escribir una expresión para el tiempo promedio de acceso a memoria, en términos de m y B.
2. ¿Qué tamaño de bloque otorga el mejor tiempo promedio de acceso a memoria?
3. Si la contención por el acceso al bus agrega 3 ciclos al tiempo de acceso a memoria principal, ¿qué tamaño de bloque otorga el mejor tiempo promedio de acceso a memoria?
4. Si el ancho del bus se cuadruplica a 128 bits, reduciendo el tiempo que toma la transferencia del bloque (dentro de la transacción en el bus) a un 25 % del valor original, ¿cuál es el tamaño de bloque óptimo?

   Asumir que como mínimo se requiere 1 ciclo para la transferencia, y no incluir los ciclos de contención mencionados en el ítem anterior.

3.15.

Para cada uno de los siguientes puntos, elegir la respuesta correcta justificando su elección (la respuesta siempre se toma como errónea si no está debidamente justificada)

1. Si el acceso a caché requiere un ciclo de reloj y cada caché miss produce un *stall* de 5 ciclos adicionales, ¿cuál de los siguientes *caché hit rates* se acerca más a lograr un acceso promedio a memoria de 2 ciclos?

a) 75 %

b) 80 %

c) 83 %

d) 86 %

e) 98 %

2. LRU es una estrategia de reemplazo en caches efectiva principalmente debido a que los programas ...

   a) ... exhiben localidad de referencias a memoria

   b) ... usualmente trabajan en regiones de memoria pequeñas

   c) ... leen datos más frecuentemente de lo que los escriben

3. Si incrementar el tamaño de bloque de un caché mejora su performance, esto se debe principalmente a que los programas ...

   a) ... exhiben localidad de referencias a memoria

   b) ... usualmente trabajan en regiones de memoria pequeñas

   c) ... leen datos más frecuentemente de lo que los escriben

4. Dado el siguiente programa:

```
int a[1000];
size_t i;
for(i=0; i<1000; i++)
        for(j=0; j<1000; j++)
                a[i]=a[i]+1;
```

   Cuando es compilado con todas las optimizaciones apagadas y corrido en un procesador con un caché de datos de 1KB, 4 words por línea, write-back y fully-associative, ¿cuál es aproximadamente el miss rate que presenta este caché?

   (asumir int = size_t = word = 32 bits)

   a) 0,0125 %

   b) 0,025 %

   c) 0,05 %

   d) 0,1 %

   e) 5 %

   f) 12,5 %

5. En un procesador de un ciclo por instrucción (CPI_{exec}) con un caché de instrucciones, el miss rate de dicho caché es del 5 %. Realizar el fetch de una línea de caché desde memoria principal toma 8 ciclos. Despreciando los caches misses de datos, ¿cuál es el CPI promedio aproximado?

   a) 0,45 %

   b) 0,714 %

   c) 1,4 %

   d) 1,8 %

   e) 2,22 %

3.16.

Se tiene una arquitectura con las siguientes características:

- Caché 4WSA de 32KB y 128 conjuntos virtualmente direccionado.
- Direcciones virtuales de 48 bits.

1. Indicar el tamaño en bits que tendrá el arreglo de Tags dentro del caché.
2. Ídem, pero cache fully-associative de 32KB y líneas del mismo tamaño.

Tener en cuenta sólo los tags, no contabilizar otros bits como valid, dirty, etc.

3.17.

Se tiene un sistema de memoria virtual con una tabla de páginas lineal y páginas de 16 bytes. El total de memoria física es de 4096 bytes (es decir, direcciones físicas de 12 bits). Sin embargo, cada proceso tiene acceso sólo a 256 bytes (i.e., direcciones virtuales de 8 bits). Un volcado de una porción física de memoria es el del cuadro 2, junto con la tabla de páginas del cuadro 3:

Physical Address	Data (bytes)															
080	2A	1B	0C	3D	5A	02	13	12	22	13	4A	0B	10	21	21	12
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	02	1B	A1	2C	11	31	22	33	11	12	14	2B	11	2B	15	13
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	02	01	03	04	11	01	01	0B	11	10	12	13	00	03	11	0B
0F0	00	00	01	10	00	00	00	00	00	00	00	00	00	00	00	00
100	1A	2B	3C	4D	5E	01	10	02	20	03	40	0A	11	22	01	10
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	01	1A	A0	2B	10	21	12	13	01	02	04	2A	01	1B	02	03
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	01	01	02	03	10	01	02	0A	10	11	02	03	00	02	10	0A
170	01	02	02	01	01	02	03	04	00	00	00	00	00	00	00	00

Cuadro 2: volcado de memoria (todos valores hexa)

1. Indicar cuál es la dirección física correspondiente a la dirección virtual 0x70.
2. ¿Cuál es el valor del byte en la dirección virtual 0xc1?
3. ¿Cuál es el valor hexadecimal del entero de 32 bits en la dirección virtual 0xec? ¿Qué suposición tiene que hacer?

3.18.

En este problema, examinaremos las tablas de paginación en procesadores con direccionamiento de 64 bit.

1. Para una computadora con direcciones virtuales de 64 bit, con una tabla de paginación plana (i.e. no jerárquica), ¿Qué tan grande es la tabla de paginación? Suponer que el tamaño de página es 4KB, que cada entrada en la tabla ocupa 8 bytes, y que el procesador permite accesos de byte a la memoria.

VPN	PPN
0	08
1	09
2	0A
3	0B
4	0C
5	0D
6	0E
7	0F
8	10
9	11
A	12
B	13
C	14
D	15
E	16
F	17

Cuadro 3: Tabla de Páginas

2. Actualmente, ciertas ISAs de 64 bit implementan sólo una parte del espacio de direcciones. Una forma de lograrlo, es segmentando el espacio en tres partes: stack, código y heap, y una tercer área, reservada.

   Además, antes de procesar la dirección, el sistema de memoria virtual usa un circuito para detectar si los 8 bits más significativos de una dirección son todos cero o todos uno. De no ser así (i.e., algunos cero y otros uno), se genera un trap para indicar que la dirección es inválida. En efecto, este esquema elimina los 7 bit más significativos de la dirección virtual, pero permite implementar un esquema compatible con diseños futuros, y con espacios de direccionamiento virtual más grandes.

   El procesador MIPS R10000 hace algo similar: como las direcciones de 64 bit son innecesariamente grandes, sólo los 44 bits menos significativos son traducidos. Esto, a su vez, permite reducir el costo del TLB, y del procesamiento de los tags en el cache. Los dos bits más significativos (63:62), permiten seleccionar entre los espacios de usuario, supervisor, y kernel. Los bits intermedios 61:44 deben ser todos uno o todos cero, dependiendo de la región particular.

   Para un procesador R10000, ¿qué tan grande será la tabla de paginación en un esquema no jerárquico? Nuevamente, suponer que cada página mide 4KB, que cada PTE ocupa 8 bytes, y que el procesador permite accesos de byte a memoria.

3. Mediante un esquema jerárquico, es posible reducir el tamaño de las tablas de paginación. Supongamos particionar los 44 bit efectivos de la direcciones virtuales, de la siguiente forma:

Figura 1: Composición de las direcciones virtuales (4KB).

   Definiendo el overhead de traducción (PTO), como el cociente entre (numerador) la cantidad de memoria física usada para almacenar las tablas de paginación, y (denominador) la cantidad de memoria física dedicada a páginas de código y datos de usuario, ¿cuál es el menor PTO posible en un esquema jerárquico de tres niveles? ¿Cuál es el mayor valor del PTO? Suponer que existe suficiente memoria física, como para que no haya swap de páginas a disco; y usar PTEs de 64 bit. Además, a los efectos del cálculo, considerar que el contenido completo de la página alocada es útil.

4. MIPS R10000 usa direcciones físicas de 40 bit. La sección de traducción de la TLB contiene el número de página física (`PFN`), un bit `V` para indicar si la entrada es válida, un bit `D` (dirty) para indicar que la página necesita ser procesada debido a que fue modificada, y tres bits de estado.

   ¿Cuál es el tamaño mínimo de la PTE, suponiendo páginas de 4KB?

Index	Longitud en bits
Top-level	
2^{nd}-level	
3^{rd}-level	

Cuadro 4: ver ejercicio 6.

5. MIPS/Linux almacena cada PTE en una palabra de 64 bit. Usando la respuesta a la última pregunta, ¿cuántos bits serán desperdiciados?

6. El siguiente comentario, extraído de una versión de Linux/MIPS, describe la jerarquía de tres niveles:

```
/*
  * Each address space has 2 4K pages as its page directory, giving
  * 1024 8 byte pointers to pmd tables. Each pmd table is a pair of
  * 4K pages, giving 1024 8 byte pointers to page tables. Each (3rd
  * level) page table is a single 4K page, giving 512 8 byte ptes.
  */
```

   Completar el cuadro 4, suponiendo páginas de 4KB.

7. Durante el diseño de un cache 4-way set associative, Raúl R. nota que el cache sufrirá un problema de aliasing homónimo: tal problema sucede cuando dos procesos usan la misma dirección virtual para acceder a lugares físicos distintos.

   Raúl entonces consulta con el licenciado Varela, quien sugiere agregar un campo PID (process id) al tag virtual. ¿Solucionará esto el problema de aliasing?

   Otro problema que surge al usar caches virtualmente indexados, y virtualmente taggeados, es la aparición de sinónimos: los mismos aparecen cuando direcciones virtuales distintas refieren al mismo lugar físico. ¿Solucionará este problema la idea del licenciado?

   Raúl piensa que otra forma de solucionar estos problemas, será usando un cache direct mapped, en vez de set associative. ¿Tiene razón?

3.19.

Considerar una máquina con direcciones virtuales de 64 bits y páginas de 4KB. Cuenta con una tabla de páginas jerárquica de tres niveles, donde los índices L1, L2 y L3 son de 12 bits cada uno. Los bits más significativos no se utilizan. Ver cuadro 5.

unused	L1 index	L2 index	L3 index	page offset
63				0

Cuadro 5: Formato de las direcciones virtuales

1. ¿Cuántas entradas hay en la tabla de páginas del nivel 1?

2. ¿Cuál es el tamaño de la parte implementada del espacio de direcciones virtuales?

3. La figura 2 muestra fragmentos del contenido de la tabla de páginas jerárquica de tres niveles (la columna a la izquierda corresponde a las direcciones físicas de cada entrada en memoria).

   Las PTEs en las tablas de nivel 1 y 2 contienen la dirección física (PA) a o el número de bloque de disco (DBN) de la tabla del siguiente nivel.

   Las PTEs de la tabla de nivel 3 contiene el número de página física (PPN) o el número de bloque de disco (DBN) de la página accedida. Las direcciones físicas son de 28 bits.

   El tamaño de las PTEs es de 4 bytes, y la dirección base (valor de root pointer) de la tabla actual es 0x0004000. Las direcciones más bajas corresponden a los índices más bajos de las

tablas de páginas. Sólo se muestran las páginas válidas. La columna “R” indica el bit de página “residente” (o presente en memoria).

Para cada dirección virtual dada a continuación, indicar la respuesta correcta y llenar el espacio en blanco si corresponde:

a) Recorrido de las tablas de paginación para la $VA = 0x0000004010110804$

1) Page table entry invalid exception
2) Page fault on page table entry
3) Page fault, disk block number _____
4) Redisent, physical address _____

b) Recorrido de las tablas de paginación para la $VA = 0x00000001113B0110$

1) Page table entry invalid exception
2) Page fault on page table entry
3) Page fault, disk block number _____
4) Redisent, physical address _____

c) Recorrido de las tablas de paginación para la $VA = 0x0000001101002CD0$

1) Page table entry invalid exception
2) Page fault on page table entry
3) Page fault, disk block number _____
4) Redisent, physical address _____

	PA/PPN/DBN	R
0x001C450	0x000100B	0
0x001C44C	0x000100D	0
0x001C448		
0x001C444	0x000100E	0
0x001C440	0x0020000	1
0x0014440	0x0110	1
0x001443C	0x0100	1
0x0014438		
0x0014434	0x0108	1
0x0014430		
0x0010050		
0x001004C		
0x0010048		
0x0010044	0x000100C	0
0x0010040	0x0014000	1
0x000C010	0x0104	1
0x000C00C		
0x000C008	0x000100A	0
0x000C004		
0x000C000	0x010C	1
0x0008410	0x0024000	1
0x000840C		
0x0008408		
0x0008404	0x000C000	1
0x0008400		
0x0004010	0x0010000	1
0x000400C		
0x0004008		
0x0004004	0x0008000	1
0x0004000	0x001C000	1

Figura 2: Volcado de memoria en la región de la tabla de páginas

4. La máquina descripta cuenta con un caché único L1, el cual es accedido en paralelo con una TLB fully-associative de 64 entradas. El caché es Direct Mapped, de 16KB, bloques de 4 words, virtually-indexed y physically-tagged. El tamaño del word es de 4 bytes.

   a) ¿Cuáles de los 64 bits de la dirección virtual son traducidos por la TLB? _____ : _____

   b) ¿Cuáles de los 64 bits de la dirección virtual se utilizan para indexar dentro del cache L1? _____ : _____

   c) ¿Cuáles de los 28 bits de la dirección física forman el cache tag? _____ : _____

5. En este sistema se previene *aliasing* de la siguiente manera: si dos direcciones virtuales mapean a la misma dirección física, se requiere que el sistema operativo haga que los page offsets de las direcciones virtuales coincidan. ¿Esto funciona? (justifique, de lo contrario la respuesta se considera automáticamente incorrecta)

3.20.

Sea un sistema de memoria virtual que permite alojar páginas de 4KB y 4MB. El sistema usa 44 bits efectivos para describir direcciones virtuales, y 40 para direcciones físicas. Las páginas de 4KB son organizadas usando una tabla jerárquica de tres niveles; y las de 4MB se organizan usando los dos primeros niveles de la misma tabla.

Para distinguir entre ambos casos, las L2 PTEs contienen información que indican si el puntero apunta a una tabla L3, o a una página de 4MB. Todas las PTEs son de 8 bytes. Ver figura 3.

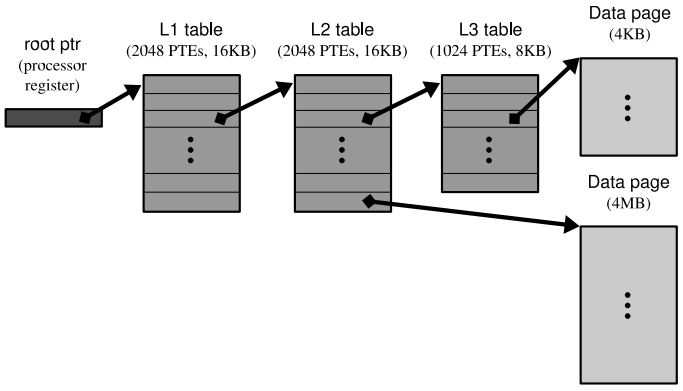

Figura 3: jerarquía de traducción de direcciones.

El procesador dispone de un TLB de datos con 64 entradas, y cada entrada puede mapearse a cualquiera de los dos tipos de página, 4KB ó 4MB. Al ocurrir un TLB miss, las tablas de traducción son recorridas por hardware, a fin de recargar el TLB. La política de reemplazo es FIFO.

En este ejercicio, evaluaremos la ejecución y utilización de memoria del siguiente programa; el mismo es usado para sumar dos arreglos, y almacenar el resultado en un tercero:

```
uint8_t A[1048576]; /* 1MB array */
uint8_t B[1048576]; /* 1MB array */
uint8_t C[1048576]; /* 1MB array */
for(int i = 0; i < 1048576; ++i)
C[i] = A[i] + B[i];
```

Suponemos, además, que los arreglos `A`, `B`, y `C` están ubicados en una región contigua de memoria física. Consideraremos, además, dos mapeos posibles:

- 4KB: los arreglos son mapeados usando 768 páginas de 4KB (es decir, cada arreglo usa 256 páginas);
- 4MB: los arreglos son mapeados usando una única página.

En las preguntas que siguen, suponer que el programa es el único proceso en el sistema, e ignorar cualquier overhead asociado a la ejecución de las instrucciones, o con el sistema operativo. Suponer que los arreglos están alineados de tal forma, que minimizan el número de PTEs involucradas.

1. El particionado de la dirección virtual, en el caso de 4KB, es: 11 bits (L1 index), 11 bits (L2 index), 10 bits (L3 index), 12 bits (page offset). Ver figura 4.

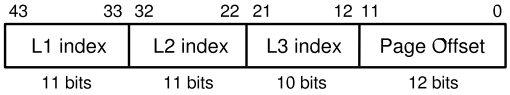

Figura 4: composición de las direcciones virtuales (4KB).

   Mostrar, en forma similar, cómo sería el particionado de una dirección virtual que mapee a una página de 4MB. Incluir el nombre y longitud en bits de todos los campos involucrados.

2. Definimos el overhead de traducción (PTO), como el cociente entre (numerador) la cantidad de memoria física usada para almacenar las tablas de paginación, y (denominador) la cantidad de memoria física dedicada a páginas de datos.

   Para el programa dado anteriormente, ¿cuál es el PTO_{4KB}? ¿cuál es el PTO_{4MB}?

3. Definimos el overhead de fragmentación (PFO), como el cociente entre (numerador) la cantidad de memoria física dedicada a páginas de datos, pero que nunca es accedida; y (denominador) cantidad de memoria física alocada a páginas de datos, accedida.

   Para el programa visto, ¿cuánto valen PFO_{4KB} y PFO_{4MB}?

4. Consideremos ahora la ejecución del programa, suponiendo que la TLB está inicialmente vacía. Para cada uno de los casos (i.e. 4KB y 4MB), ¿cuántos TLB misses ocurren, y cuántos accesos a memoria (PTEs) son necesarios para recargar la TLB?

5. ¿Cuál de los siguientes números es el mejor candidato para estimar (orden de magnitud) el speedup? $SU = \{1{,}01; 10; 1000; 1000000\}$. Elegir uno, explicando brevemente tu respuesta. Tomar el caso de 4KB como referencia.