Problema 1. Tipos de máquinas

Dada la siguiente expresión donde R, A,B,C y D son variables globales:

```
R = C-(A-B)/(C-D)
```

a) **Programa** una secuencia de código que la evalue en una arquitectura de tipo pila con las siguientes operaciones:

```
add/sub/mul/div  #pila[top+1]=pila[top] op pila[top+1]; top=top+1
push @           #top=top-1; pila[top]=M[@]
pop @            #M[@]=pila[top]; top=top+1
```

b) **Programa** una secuencia de código que la evalue en una arquitectura de tipo acumulador con las siguientes operaciones:

```
add/sub/mul/div @  #ACC=ACC op M[@]
load @             #ACC=M[@]
store @            #M[@]=ACC
```

Problema 2. RISC-CISC

Un cachivache electrónico (llamado i-Cachivache) ejecuta repetidamente un programa que no puede tardar más de 2,5 segundos. Para ello el i-Cachivache incorpora un procesador CISC registro-memoria. Al ejecutar el programa compilado para dicho procesador hemos obtenido los siguientes datos:

- 10^9 instrucciones dinámicas ejecutadas
- CPI 2,5 ciclos/instrucción
- el 30% de las instrucciones tienen un operando en memoria
- el 10% de las instrucciones tienen dos operandos en memoria
- el 70% de los accesos a memoria usan el modo base+desplazamiento
- el 20% de los desplazamientos se codifican con más de 16 bits
- el 20% de los accesos a memoria usan el modo registro indirecto (equivalente a base + desplazamiento cero)
- el 20% de las instrucciones tienen un operando inmediato
- el 15% de los inmediatos se codifican con más de 16 bits.

a) ¿Cuantos accesos a datos se realizan durante la ejecución del programa?

b) ¿A que frecuencia debemos hacer funcionar el procesador para que el programa se ejecute en el tiempo previsto?

En la segunda generación del cachivache (el i-Cachivache 2) deseamos sustituir el procesador CISC por un RISC que esperamos tenga un menor consumo y aumente la duración de la batería. Para ello disponemos de un traductor que traduce del lenguaje máquina CISC al RISC. Al traducir el programa hemos obtenido que:

- El 90% de las instrucciones CISC se han podido mapear sobre una instrucción RISC equivalente. El 10% restante ha necesitado 2 instrucciones RISC.
- Los accesos a memoria han necesitado una instrucción adicional de Load/Store.
- Los accesos a memoria que no se han podido implementar usando el modo base+desplazamiento han necesitado otra instrucción adicional para calcular la dirección.
- Los load/store con desplazamientos de más de 16 bits han necesitado una instrucción adicional ya que el RISC sólo dispone de 16 bits para codificarlos.
- Los inmediatos que necesitan más de 16 bits también han necesitado una instrucción adicional ya que los inmediatos también están limitados a 16 bits.

c) ¿Cuantas instrucciones dinámicas ejecutará el procesador RISC?

Al ejecutar el programa traducido en el procesador RISC hemos obtenido un CPI de 1,2 ciclos/instrucción.

d) ¿A que frecuencia deberíamos hacer funcionar el procesador RISC?

El procesador RISC es ligeramente más pequeño que el CISC por lo que su intensidad de fuga y su carga capacitiva son también menores, 10 A y 50 nF en el CISC por solo 8 A y 40 nF para el RISC. Ambos procesadores funcionan con una tensión de alimentación de 1 V. En ambos procesadores la potencia de cortocircuito es despreciable, por lo que solo tendremos en cuenta la potencia de fuga y la de conmutación.

e) Calcular la energía consumida por ambos procesadores al ejecutar el programa.

f) Calcular la ganancia en duración de batería del RISC sobre el CISC

Al cabo de un tiempo se dispone de un compilador que nos permite compilar el programa directamente para el procesador RISC. Al ejecutar el programa compilado directamente para el RISC, vemos que el número de instrucciones dinámicas se ha reducido a $1{,}5 \times 10^9$ instrucciones, pero el CPI ha aumentado a 1,3 ciclos/instrucción. Esto nos permitirá sacar al mercado el i-Cachivache 3 que, a parte de un nuevo diseño más "cool" de la carcasa, es idéntico al i-Cachivache 2, pero con la nueva versión del código).

g) ¿A que frecuencia debería funcionar la CPU RISC con el nuevo programa compilado?

h) ¿Cual será la ganancia en duración de batería con el nuevo código?

Problema 3. Microoperaciones

Los actuales procesadores CISC de la familia x86 (tanto de Intel como AMD) traducen internamente las instrucciones de lenguaje máquina x86 a microperaciones (que denominan uops). En una implementación de x86 tenemos las siguientes uops:

Tipo	Descripción	Ejemplos
LOAD	Rd <- M[Rb+ Ri*s+dspl]	LOAD %r1 <- M[%ebx+%edx*4+36]
STORE	M[Rb+ Ri*s+dspl] <- Rf	STORE M[%ebx+%edx*4+36] <- %r3
MOV	Rd <- Rf1 Rd <- inmediato	MOVL %eax <- $3 MOVL %r1 <- %ebx
Aritmética	Rd <- Rf1 (op) Rf2 Rd <- Rf1 (op) inmediato	ADDL %r1 <- %r2 + %eax IMULL %eax <- %r4 * $20
Comparación	flags <- Rf1 - Rf2 flags <- Rf1 - inmediato	CMPL %eax , %r5 CMPL %eax , $100
Salto	mismos saltos que x86	JGE loop

Solo las instrucciones de LOAD y STORE pueden acceder a memoria, el resto sólo pueden tener registros o inmediatos como operandos. Ademas de los registros visibles en x86 disponemos de registros adicionales para almacenar valores intermedios (%r1 ... %r8). Los modos de direccionamiento de LOAD y STORE son los mismos que x86.

a) **Traducir** las siguiente secuencia de instrucciones en x86 a la correspondiente secuencia de microoperaciones

```
        movl $0, %ecx
loop:   cmpl $1000000, %ecx
        jge fin
        movl x, %eax
        imull V(,ecx,4), %eax
        addl %eax, suma
        incl %ecx
        jmp loop
fin:
```

b) **Calcular** cuantas instrucciones dinámicas y cuantas uops dinámicas se ejecutan

Al ejecutar este código nuestra CPU obtiene un UPC (uops por ciclo) de 1,3

c) **Calcular** el CPI de este código

Suponiendo una frecuencia de 3GHz

d) **Calcular** el tiempo de ejecución de este código

Supongamos que las instrucciones x86 ocupan:

- 1 byte de código de operación (OpCode)
- 1 byte adicional si tiene 2 o más operandos (Modo), las de 1 sólo operando lo codifican dentro del OpCode
- 4 bytes adicionales si alguno de los operandos es un inmediato (incluidos los destinos de los saltos)
- 1 byte adicional (SIB) si accede a memoria
- 4 bytes adicionales si el modo de direccionamiento incluye desplazamiento

Las uops ocupan todas 6 bytes.

e) **Calcular** el tamaño del código x86 y el que ocuparían las uops equivalentes.

La familia *Sandy Bridge* de Intel (2011) incluye, ademas de una cache de instrucciones de nivel 1, una cache de micro-ops (que denominan de nivel 0) donde se guardan las uops que ya han sido traducidas para que en caso de hit no sea necesario descodificar y traducir las instrucciones x86.

f) **Calcular** el numero de bytes leidos y el ancho de banda efectivo de la cache de instrucciones al ejecutar este código (sin uop cache)

g) **Calcular** el numero de bytes leidos y el ancho de banda efectivo de la cache de uops al ejecutar este código (con uop cache)

Una de las ventajas de la cache de micro-uops es que se produce un ahorro de energía ya que no es necesario descodificar todas las instrucciones dinámicas. En nuestro procesador, un acceso, tanto a la cache de instrucciones como a la cache de uops, consume 1 nJ (nanoJulio) por byte leido. Descodificar una instrucción consume 10 nJ. El código ejemplo cabe tanto a la cache de instrucciones como en la cache de uops, que inicialmente están vacias.

h) **Calcular** la energía consumida por nuestro programa en un procesador sin cache de uops (las instrucciones se leen de la cache de instrucciones y se decodifican cada vez) , en uno con cache de uops (las instrucciones se leeen ya decodificadas de la cache de uops, la cache de datos y el decodificador solo se usan cuando es fallo en la cache de uops) y la ganancia en consumo de energía debida a la cache de uops durante la búsqueda y descodificación de instrucciones.